The PUB

PINTofALE Users' Book

A guide for the use of the
Package for INTeractive Analysis of Line Emission

Jeremy Drake
Vinay Kashyap
LiWei Lin

`pintofale@head.cfa.harvard.edu`

Version 2.43
1 Mar 2007

Contents

New in PINTofALE 1.0

The 1.0 release was the first publicly available version of `PINTofALE`. Previous versions of the software that users might have seen in demonstrations or advertised in papers and posters were informally maintained and modified (VLK) or hacked (JJD) to suit particular tasks. The current coding is (almost) entirely written by VLK, adopting a relatively uniform programming style. Otherwise, the code design has changed relatively little since its inception, other than in details of what is included in, eg, the ID structure, and improvements in GUI's, addition of routines etc.

Being the first release, this `PINTofALE` version is bound to have some residual problems and bugs. However, we think the software is sufficiently well-developed that any such problems will be somewhat minor in nature. All the routines have been tested and used by ourselves to some extent. However, there are many options to some of the procedures and functions, and some of these, originally programmed for convenience, have inevitably been less used than others. We strongly encourage you to contact us if you encounter any problem or unexplained behavior with `PINTofALE`. Being a modular IDL system, simple bug fixes can be implemented immediately, and new routines posted on the website. We also encourage feedback if any procedure and function descriptions in the `PUB` Appendix and on-line, or any of the `PUB` itself, is found to be lacking or unclear. It has been a major effort and undertaking to produce this documentation and decriptions of all the routines, even in this current rather limited form, but we are eager to improve it where we can.

New in PINTofALE 1.5

After numerous incremental patches incorporating bug fixes and cosmetic improvements, we are releasing a new version that has some significant changes from the original release, as version 1.5. The most significant changes are listed below:

1. Improved initialization and color handling
2. Compatibility with CHIANTI v4
3. Access to APED via ATOMDB
4. Improved line-fitting capabilities, including ability to use MPFIT

New in PINTofALE 2.0

Since the last major release (v1.5, 9-Oct-2002), we have continued to make improvements and bug fixes to the previously released code, as well as add new routines to the mix. These changes are too numerous to mention, but the biggest improvement has been that we have brought the code and the calculated emissivities in sync with the latest version of CHIANTI (v4.2). In addition, we have added significant new functionality to the emission measure analysis – this new version now contains an implementation of a Markov-Chain Monte Carlo algorithm to derive stellar coronal differential emission measures from line flux measurements of high-resolution spectral data.

Check the website

`http://hea-www.harvard.edu/PINTofALE/`

for new developments and routines that have become available since this release. Due to the large amount of time it takes to produce documentation, this PINTofALE Users' Book will only be revised at the time of a major new release.

Future Developments

The main directions for future development in `PINTofALE` include the following:

- Streamlining emissivity database to reduce the storage requirements. This will likely be achieved without altering the output of database reading routines like `RD_LINE` (except for the [currently unused] `SRC` extension, which will likely be changed to a string array). The format and number of the database files, however, is expected to change.
- Improved graphical support, especially for positioning line ID labels, setting colors, etc.
- A complete inclusion of the Astrophysical Plasma Emission Database (APED)
- Treatment of many weak lines using a pseudo-continuum.
- Inclusion of more instrument-specific "calibration" information, especially for XMM-*NEWTON*, and improvements in the *Chandra*-specific data.
- More generalised spectral fitting including continua and "table models".
- Limited support for photoionization models and modelling.
- More examples of the use of `PINTofALE` applied to various analysis tasks, including absorption line analysis.

Please contact us if you require any routines, developments that are not currently included in `PINTofALE` or this list, or modifications and improvements to existing `PINTofALE` routines. If we can, we will be happy to accomodate these suggestions.

.

Chapter 1

Introduction

Anyone wishing to analyze EUV and X-ray spectra traditionally had a choice of how to proceed: gather the necessary atomic and ancillary data oneself and perform analyses “by hand”, or use an existing “black box” compilation of data hard-coded into a program with a more fixed functional capability. Excellent and extremely useful “black box”-type programs such as Sherpa (Freeman et al. 2000), XSPEC (Arnaud 1996), and SPEX (Kaastra et al. 1996) are available. However, they were developed with the idea in mind of statistical fitting of spectral models to low resolution X-ray spectra, and are often not sufficiently flexible to easily perform analysis tasks one would like to apply to high resolution spectra, or to analyses that depart from the global model fitting type. Unfortunately, the alternative approach of doing the analysis “by hand” can be extremely time-consuming, especially for the case of high resolution spectra from, e.g., EUVE, *Chandra*, and XMM-Newton: identifying the lines in complex spectra, measuring line emission/absorption profiles, deblending the contributions to lines of interest from other species, accounting for the temperature and density sensitivity of lines and continua, including effects of interstellar absorption and instrument effective areas, etc.

`PINTofALE` (Kashyap & Drake 2000) was motivated by the large amount of book-keeping, computation and iterative interaction that is required between the researcher and observational and atomic data in order to derive astrophysical results outside of the “black box” global fitting approach. The aim was to develop a more flexible system for analyzing spectra and one that offers more transparent access to atomic data so that the user has complete control over what data to use for a given analysis. We chose the *Interactive Data Language*$^{\mathrm{TM}}$ (IDL) as the basis on which to build. Even though it is a proprietary system, the IDL language was a natural choice for such a development because of its underlying infrastructure, plotting and GUI capabilities, as well as its existing popularity in the astrophysical community.

New S-lang based functionalities are being built into CIAO/Sherpa to analyze line emission (e.g., GUIDE), based on concepts developed and demonstrated with `PINTofALE` (cf. §3.1.5).

1.1 Overview

`PINTofALE` is based on a modular set of IDL tools that were written to interact with a formatted atomic database and with observational data. There are 3 basic components to the system:

1. routines to handle externally compiled atomic databases such as `CHIANTI` and `SPEX` (`APED` will be included in a future release);
2. routines to access the database of lines, continua, and ion balance compiled from the external databases; and
3. routines to carry out various analysis tasks such as flux measurement, feature identification, emission measure estimation, DEM reconstruction, density estimation, etc.

This structure is illustrated in Figure 1.1. Atomic data from external compilations are recast into a uniform format, which are then read into the IDL environment and manipulated conditional on source parameters and observational data. The diagram is intended to represent the interactive and iterative nature of this process.

Strictly speaking, the full scope of `PINTofALE` is currently limited in application by some of its design characteristics and by the limitations of the atomic data we are able to include. Broadly, this corresponds to optically-thin emission in the 1-1500 Å range from coronal plasma in collisional equilibrium. However, many routines in `PINTofALE` are completely general and can be applied to any analysis tasks. Anyone wanting to fit and deblend spectral lines, graphically interact with a spectral window, or add identification labels to a spectrum (or any vector plot) interactively, can find `PINTofALE` useful.

1.1.1 Silly Name

The often denounced and somewhat silly `PINTofALE` acronym was a spur-of-the-moment monicker thought up for the High Energy Astrophysics Center

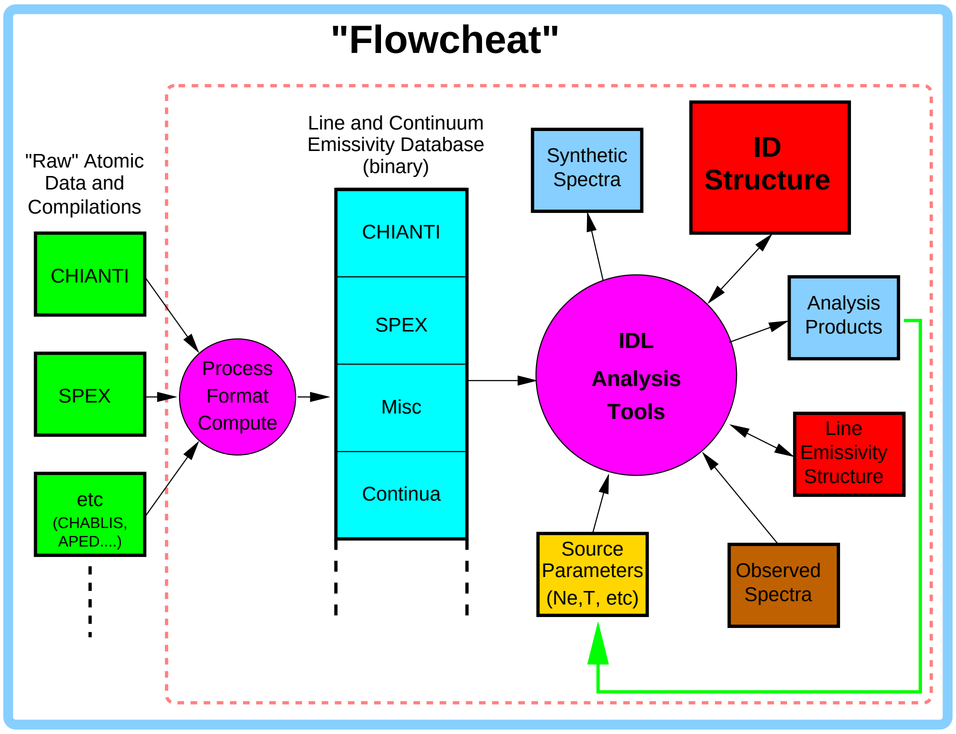

Figure 1.1: The structure of `PINTofALE`. Not exactly a flowchart, this figure illustrates the basic design of ths system: atomic data from external compilations are recast into a uniform format, which can then read into the IDL environment and manipulated according to source parameters and observational data.

Coordination (HEADCC) meeting of December 1999 when the software was first publicly demonstrated. The new atomic databases of the time were CHIANTI (K.Dere et al) and CHABLIS (M.Laming), so it seemed that alcoholic beverages were an appropriate way to go.

1.1.2 How to contact us

We would like to hear about your experiences with `PINTofALE`, especially if you have found problems, or if you have suggestions or routines that you think could improve `PINTofALE`. To submit suggestions for improvements, your own routines you think would be of value for including in a subsequent release, bug reports, or if you have any problems with `PINTofALE`, or want to be added to our electronic mailing list for updates and new releases, please email us at

`pintofale@head-cfa.harvard.edu`

Up to date information is available online at

`http://hea-www.harvard.edu/PINTofALE/`

1.1.3 Acknowledgements

We express warm thanks to Rebecca McMullen, Olivia Johnson, Antonio Maggio, Pete Ratzlaff, and Miriam Krauss for many helpful comments during development. `PINTofALE` was largely developed in our "spare" time; JJD was also supported by NASA contract NAS8-39073 to the *Chandra X-ray Center*, and VLK acknowledges support from NASA grants NAG5-3173, NAG5-6775, NAG5-3831, NAG5-3196, NAG5-7226 and NAG5-3195. This release was made possible through a much needed grant from the NASA Applied Information Systems Research Program.

Chapter 2

Installation

Instructions for download and installation are included in the files `http://hea-www.harvard.edu/PINTofALE/doc/installation.txt` and `http://hea-www.harvard.edu/PINTofALE/doc/LDB_CHIANTI.howto`, which are included below. If you did not obtain this guide from the website at the same time as the distribution tar files, it is possible that these instructions are for an older `PINTofALE` version than the current release. Instructions for download will be posted on the website, and up-to-date installation instructions will always be included in the above mentioned `installation.txt` file. Most of the tasks are easily performed by cutting and pasting these instructions; in some cases directory pathnames would have to be changed to suit local conditions.

```
===================================================================
PINTofALE INSTALLATION INSTRUCTIONS                       20nov2013

The minimum requirements to use PINTofALE are:
o Disk space of approx 500 MB (IDL procedures, SPEX and CHIANTI
  constant-density line emissivity databases, CIE continuum
  emissivity database)
o IDL version 5 or greater
o IDLASTRO library routines

Having CHIANTI(v7+) is also useful, especially because PINTofALE
primarily uses the ion balance files compiled by CHIANTI.

To install PINTofALE:

0. If all you need are the IDL procedures, download and unpack
http://hea-www.harvard.edu/PINTofALE/PoA_pro_current.tar.gz
```

```
   and add the location to IDL's !PATH.  That's it.

   To get the emissivities and set it up to use for atomic data
   analysis, see below.

1. Obtain the tar files containing the necessary IDL routines,
   the continuum emissivity database, and sundry documentation,
   from        http://hea-www.harvard.edu/PINTofALE/

   Either:

   1.0 All the components at once:
        http://hea-www.harvard.edu/PINTofALE/PoA_current.tar.gz

   OR each component separately:

   1.1 Routines:
        http://hea-www.harvard.edu/PINTofALE/PoA_pro_current.tar.gz
   1.2 Continuum Emissivities:
        http://hea-www.harvard.edu/PINTofALE/PoA_CIE_current.tar.gz
   1.3 SPEX Emissivities:
        http://hea-www.harvard.edu/PINTofALE/PoA_spex_current.tar.gz
   1.4 Documentation:
        http://hea-www.harvard.edu/PINTofALE/PoA_doc_current.tar.gz

   In addition, you may also download a set of line emissivities
   derived from the
   a] CHIANTI dataset
http://hea-www.harvard.edu/PINTofALE/PoA_chianti_current.tar.gz
   b] ATOMDB dataset
http://hea-www.harvard.edu/PINTofALE/PoA_atomdb_current.tar.gz
   (these are not included in the full distribution in [1.0] above)

   Note that emissivities consist of binary files that were written
   on a Sun.  If these files are not read in properly on your
   architecture, please contact us.

   Also note that any errors in the resulting emissivities are
   our responsibility, and not that of the CHIANTI, SPEX, and
   APEC software teams.

2. Unpack the downloaded tar files.  Assuming that PINTofALE
   will be installed in the directory /foo/bar/, type

        cd /foo/bar/
```

```
gunzip -c PoA_current.tar.gz | tar xvf -
```

or

```
gunzip -c PoA_pro_current.tar.gz | tar xvf -
gunzip -c PoA_CIE_current.tar.gz | tar xvf -
gunzip -c PoA_spex_current.tar.gz | tar xvf -
gunzip -c PoA_doc_current.tar.gz | tar xvf -
```

2a. If earlier installations exist, then please delete the following files, if they exist:

```
if ( -f PINTofALE/pro/marfrmf.pro ) /bin/rm PINTofALE/pro/marfrmf.pro
if ( -f PINTofALE/pro/util/setcolor.pro ) /bin/rm PINTofALE/pro/util/setcolor.pro
if ( -f PINTofALE/pro/util/which.pro ) /bin/rm PINTofALE/pro/util/which.pro
```

3. Edit the file

   `/foo/bar/PINTofALE/ardb/initale.par`

   and set the hardcoded values of !LDBDIR, !CDBDIR, and !CHIDIR to reflect the local environment. If the names do not begin with a "/" or a "$", then !TOPDIR is automatically prepended by the initialization script to complete the path.

   NOTES:
   1. !CHIDIR must point to the directory containing the CHIANTI dataset downloaded as described in step 6 below.
      (e.g., !CHIDIR='/foo/bar/CHIANTI/dbase/')

      WARNING: Prior to v4, the CHIANTI distribution did not include the "dbase" subdirectory. The current version of PINTofALE assumes and requires the existence of this subdirectory.

   2. !LDBDIR must point to the line emissivities that have been recast in the PINTofALE compatible format, either the SPEX emissivities of
      http://hea-www.harvard.edu/PINTofALE/PoA_spex_current.tar.gz
      (e.g., !LDBDIR = 'emissivity/spex')
      or the CHIANTI emissivities obtained as described in
      http://hea-www.harvard.edu/PINTofALE/doc/LDB_CHIANTI.howto
      (e.g., !LDBDIR = 'emissivity/chianti')
      or the APED emissivities obtained as described in
      http://hea-www.harvard.edu/PINTofALE/doc/LDB_ATOMDB.howto
      (e.g., !LDBDIR = 'emissivity/aped')

```
   3. !CDBDIR must point to the continuum emissivities in the
      PINTofALE format
        http://hea-www.harvard.edu/PINTofALE/PoA_CIE_current.tar.gz
      (e.g., !CDBDIR = 'emissivity/cont')

   4. !TOPDIR is set automatically when the initale script is run.
      The environment variables $PINTofALE, $PoA, $SCARDIR, and
      $SCAR are looked to first in that order, and if none of them
      are defined, then the path is determined by looking at the
      location of the initialization script.  The default value is
        !TOPDIR = '/foo/bar/PINTofALE'

4. The following is strictly speaking not necessary (see point 4a
   below), but is detailed here for convenience.

   It is advantageous to have IDL's system variable !PATH to contain
   the path to the PINTofALE installation.  This may be done in any
   number of ways depending on your local setup.  One possible method
   is to include the following (or equivalent) in your $HOME/.cshrc

      setenv PINTofALE /foo/bar/PINTofALE
      if ( $?IDL_PATH ) setenv IDL_PATH +$PINTofALE/pro\:${IDL_PATH}
      if (! $?IDL_PATH ) setenv IDL_PATH +$PINTofALE/pro

   At this point, PINTofALE is installed and available within IDL.

4a. Even if IDL_PATH is not set and PINTofALE is not available
   within IDL a priori, it may be fully initialized by running
   the script initale.pro -- this script figures out where it is
   located and automatically adds PINTofALE to IDL's !PATH variable.
   For such a ``cold start'' initialization, type:

IDL> .run /foo/bar/PINTofALE/pro/scrypt/initale

5. PINTofALE relies on many calls to the IDLASTRO and ROSATLIB
   library of IDL routines.  If they are not installed, then
   obtain them from
        http://idlastro.gsfc.nasa.gov/homepage.html (IDLASTRO)
        ftp://legacy.gsfc.nasa.gov/rosat/software/idl/ (ROSATLIB)
   and install them as directed.

6. It is highly recommended that CHIANTI (v7) be installed.
   Download it from

   http://www.chiantidatabase.org/
```

By default, PINTofALE uses ion-balance data from CHIANTI. If CHIANTI is not installed, users will have to obtain files formatted in a similar manner from elsewhere.

PLEASE NOTE: Any errors in PINTofALE computations carried out using the CHIANTI database are our responsibility and not that of the CHIANTI software team.

7. Build the CHIANTI line database as described in

    /foo/bar/PINTofALE/doc/LDB_CHIANTI.howto

   This version of PINTofALE is designed for use with v6 of CHIANTI.

   We have not supplied a tar file containing the CHIANTI emissivity database with the general release because it takes up 1 GB uncompressed, and will make the downloads slow and unwieldy (even a gunzipped file will require a correspondingly large available disk space), and because users have the choice of making a smaller database if they wish by following the instructions in the above file.

   However, for convenience, we have also provided a tar file containing the CHIANTI line emissivities computed at various densities and compressed to include only locally strong lines, in
   http://hea-www.harvard.edu/PINTofALE/PoA_chianti_current.tar.gz

8. Build the ATOMDB line database as described in

/foo/bar/PINTofALE/doc/LDB_ATOMDB.howto

   This version of PINTofALE is designed for use with v1.* of ATOMDB, which is based on APED.

8. Test the installation by following the example threads
http://hea-www.harvard.edu/PINTofALE/doc/EXAMPLE_JJD.html
or http://hea-www.harvard.edu/PINTofALE/doc/EXAMPLE_VLK.html

   or by running the example program(s) in the directory
/foo/bar/PINTofALE/pro/esempio/.

HOW TO CREATE THE CHIANTI LINE EMISSIVITY DATABASE 01Aug2010

```
NOTES:
a. This version of PINTofALE is designed for use with CHIANTI v6
b. For convenience, we have also provided a tar file containing the
   CHIANTI line emissivities computed at various densities, in
        http://hea-www.harvard.edu/PINTofALE/PoA_chianti_2.0.tar.gz
   The following instructions should be used if you prefer to use a
   different grid, or need the emissivities computed at constant
   pressure, etc.

1. Obtain the CHIANTI dataset from

   http://www.chiantidatabase.org/

   and install it, say at /foo/bar/CHIANTI/

2. Start IDL and initialize the variables

        WMN=0.0         ;minimum wavelength [Ang] in output database
        WMX=3000.0      ;maximum wavelength [Ang] in output database
        CHIDIR='/foo/bar/CHIANTI/dbase/'
                        ;full path to CHIANTI installation,
; _including_ the "dbase"
        OUTDIR='/foo/bar/PINTofALE/emissivity/chianti'
                        ;full path to output database location,
                        ; _without_ a trailing "/"

        NE_MIN=8.0      ;minimum log10 density [cm^-3] in database
        NE_MAX=15.0     ;maximum log10 density [cm^-3] in database
        D_NE=1.0        ;grid resolution in log10 electron density

        P_MIN=13.0      ;minimum log10 pressure [cm^-3 K] in database
        P_MAX=20.0      ;maximum log10 pressure [cm^-3 K] in database
        D_P=1.0
;grid resolution in log10 pressure (set to -1
                        ;to avoid making constant pressure database)

   These variables set up the IDL script that will extract line
   emissivities from CHIANTI and write them out in a fast-access
   form.  Two directories are created, one containing emissivities
   calculated at constant densities ("chiantiD") and one at constant
   pressures ("chianti").  Since each of these databases take up 176
   MB, in order to save space it may be advantageous to avoid making
   the constant-pressure database.  This may be achieved simply by
   setting D_P=-1.0 in the above initialization.
```

3. Run the script MK_LCOOL_CHIANTI

```
.run mk_lcool_chianti
```

This will take a few hours to complete, but does not need any intervention. The CHIANTI dataset is read in and emissivities are computed for each line at a variety of densities, pressures, and temperatures, which are then saved in a PINTofALE readable format.

4. To compare PINTofALE calculations with CHIANTI's over some wavelength range of interest, use the example script chipoacom. Start IDL afresh,

```
IDL> .run initale
PoA> v_OUTPS = 'chi42poa.ps'
PoA> .run chipoacom
```

The output in the postscript file will show the spectra calculated by PINTofALE and CHIANTI, followed by the fractional differences between the two. If all goes well, you should see only horizontal green line at Y=0 in the fractional difference plots.

5. CHIANTI has grown considerably over time, and now the full database in PINTofALE takes up more than a gigabyte. This database can be compressed to include only locally strong lines -- i.e., lines which are within a set threshold (e.g., 1%) of the strongest lines in the vicinity (e.g., with 1AA). The example script chiompress illustrates how to extract such lines and write them out to a smaller database.

```
    IDL> .run initale
PoA> chifulldir=!LDBDIR
PoA> outdir='/foo/bar/emissivity/chiompress'
PoA> hwdth=1 ;moving window half-width [Angstrom]
PoA> dynrng=1e2 ;dynamic range threshold within window
PoA> .run chiompress
```

This will create the shrunken line database directory outdirD/ and a postscript file that displays which lines were kept and which were not, for each element.

NOTE: Any errors in these computations are due to us, and are not the responsibility of the CHIANTI software team.

You should now have a complete installation of `PINTofALE`. In the near future, we will also be releasing routines to compile a `PINTofALE` format emissivity database based on the APED/APEC release (`http://hea-www.harvard.edu/APEC`).

Chapter 3

About `PINTofALE`

This `PINTofALE` manual is still incomplete. We will attempt to add (improve) documentation on the use of `PINTofALE` over the next few months. The following represents a bare bones description of some of the more important aspects of the package. The individual routines, together with cursory descriptions and calling instructions, are listed in the Appendix B.

3.1 Structural Details

The gross structure of `PINTofALE` illustrating where the emissivity database, its formatting, and the rest of the IDL tools fit in can be seen from the flowcheat in Figure 1.1. Why compute uniform format emissivity databases when, e.g., the CHIANTI release already includes routines to compute synthetic spectra and line emissivities? There are two reasons: (1) we would like to be able to handle data from a variety of different sources but with minimal effort required from the user to switch between and to mix databases; (2) computing things such as the predicted count rate for a source spectrum seen by a broad-band instrument like the ROSAT PSPC, or a high resolution spectrum in the broad-band *Chandra* LETG (covering 1.2-170 Å), would entail a very large set of statistical equilibrium computations including all the ions and transitions that enter into the bandpass if performed "from scratch".

The IDL tools themselves are comprised of *procedures*, *functions* and *scripts*.

3.1.1 Units

`PINTofALE` assumes cgs units, except where otherwise noted. One important exception is wavelength units, for which Å are used. Some routines can accept the keV energy unit by setting a keyword `ikev`; this was added *ad hoc* and was not included in all routines that accept a wavelength as an input or keyword vector. The "useful constants" initialization routine `inicon.pro` that sets the values of various fundamental conversions (e.g. the atomic mass unit (amu) in grams) and other useful values (e.g. solar radius in cm), may depart from cgs units in some cases.

The Differential Emission Measure (DEM) is generally assumed to be defined in the form

$$DEM(T) = n_e^2 \frac{d\,h}{d\ln T} \quad \mathrm{cm}^{-5}\,, \tag{3.1}$$

where n_e is the coronal electron density [cm^{-3}], T are the temperatures [K], and h is the depth of the emitting region [cm]. Substituting the emission volume $d\,V$ for $d\,h$, or including the distance to, (d) or extent (R_*) of, the source as in

$$n_e^2 \frac{d\,V}{d\ln T} \quad \mathrm{cm}^{-3}\,, \tag{3.2}$$

which is used, for example, in the script `MAKE_SPECTRUM`,

$$\frac{1}{4\pi d^2} n_e^2 \frac{d\,V}{d\ln T} \quad \mathrm{cm}^{-5}\,,\mathrm{or} \tag{3.3}$$

$$\frac{R_*^2}{d^2} n_e^2 \frac{d\,V}{d\ln T} \quad \mathrm{cm}^{-3}\,,\mathrm{etc.} \tag{3.4}$$

are also OK.

It is up to the user to make sure that the units make sense. The $\frac{d}{d\ln T}$ part, however, is crucial. If you have Emission Measure distributions obtained from elsewhere, make sure that it is first converted to the above scale.

The abundance A_X of an element X is assumed to be the fraction of the number density of the element relative to the number density of H,

$$A_X = \frac{n_X}{n_H} \quad , \tag{3.5}$$

and metallicity is assumed to be a multiplicative factor applied to all elements with $Z > 2$. All `PINTofALE` routines will accept this. In addition, some routines such as `getabund()`, `rdabund()`, and `setabund()` can also handle abundances and metallicities in the log scaling.

3.1.2 `PINTofALE` Directory Structure

All IDL code is contained in the directory `.../PINTofALE/pro`. Within this directory are subdirectories that contain:

`esempio`: example programs,

`scrypt`: scripts,

`mkemis`: routines used to make and maintain emissivity databases,

`fitting`: routines used in fitting profiles to line features,

`util`: general utility routines used by `PINTofALE`, and

`external`: routines from other places that are used by `PINTofALE` with no or minimal modifications (these do not include IDLASTRO and ROSATLIB routines).

Various files that are used by the routines (e.g., the in situ help file for `fitlines()`) are in the **a**nalysis **r**eference **d**ata**b**ase directory `PINTofALE/ardb`. This directory also contains some files with suffix `.par`, which are one-line calling sequences for the appropriate routine. The best way to use them is to copy them to your working directory and edit the names of the variables such that they reflect your existing environment, and run them as an "@" script. Here, for example, is one sequence (a detailed walkthrough using them can be found in §4.1.2):

```
        .run initale
; if INITALE fails, try
;@PoA_constructor first
        !WMAX=300.
        @/foo/bar/PINTofALE/ardb/rd_line.par
        @/foo/bar/PINTofALE/ardb/fold_ioneq.par
        @/foo/bar/PINTofALE/ardb/lineflx+ismtau.par
```

Which reads in the line emissivities, folds in ion balance, and computes ISM attenuated fluxes for a flat DEM. The results of the above set of operations may be plotted as below (for O, Ne, and Fe):

```
        plot,abs(rd_line_wvl),lineflx_flx,psym=1,$
          yr=[1e-3,20],/ys,/yl,xr=[7.,300.],/xs,/xl,/nodata,$
          xtitle='Wavelength ['+!AA+']',$
          ytitle='Line flux [ph s!u-1!n cm!u-2!n]',$
          title='line fluxes' & stample
        oz08=where(rd_line_Z eq 08) & setkolor,'blue',1
        oz10=where(rd_line_Z eq 10) & setkolor,'green',2
```

```
oz26=where(rd_line_Z eq 26) & setkolor,'red',3
oplot,abs(rd_line_wvl[oz08]),lineflx_flx[oz08],psym=1,color=1
oplot,abs(rd_line_wvl[oz10]),lineflx_flx[oz10],psym=1,color=2
oplot,abs(rd_line_wvl[oz26]),lineflx_flx[oz26],psym=1,color=3
xyouts,2.*1.4^1,10,'O',color=1,chars=1.7,charthick=2
xyouts,2.*1.4^2,10,'Ne',color=2,chars=1.7,charthick=2
xyouts,2.*1.4^3,10,'Fe',color=3,chars=1.7,charthick=2
```

All other documentation, including an example walk-through, detailed help on identifying lines, fitting line profiles, etc. can be found in the directory `PINTofALE/doc`.

3.1.3 Keywords, Keywords, Keywords and `.par` Files

The flexibility of `IDL` in allowing unlimited keywords, passed through multiple subroutine levels, is a double edged sword. While it makes the procedures and functions versatile and flexible, it also has the side effect of these routines accruing so many added keywords and options that they become more complicated to call. `PINTofALE` is designed to be a modular system as much as possible, but it was also developed over a long period of time from just a small collection of tools, without the benefit of hindsight that it would grow to its current dimensions. Consequently, as development proceeded, there was a tendency for "keyword creep" as features were added to routines to enable them to be more versatile, or to enable the output of one routine to be used as input to another, with a minimum of command line manipulation in between. On the command line, the keywords and variables needed by a routine can always be found by typing, for a function,

```
help,function_name()
```

or for a `PINTofALE` procedure by just entering the procedure name with no arguments

```
procedure_name
```

There are one or two exceptions to the first of these regarding `PINTofALE` functions; e.g., `READ_LN()` will actually start reading the database if called in this way. To further aid in understanding what keywords are needed for any particular routine, they are all listed in Appendix B, together with brief routine descriptions.

One major help in running `PINTofALE` routines that can have calls involving several arguments and keywords are the `.par` files, to be found in `.../PINTofALE/ardb/`. These files have been written for some of the more commonly used `PINTofALE` components, and contain all the arguments and keywords necessary for calling, with default variable names and system variables used where possible and convenient. The `.par` files are treated in more detail in the example sections in 4.1.

3.1.4 Initialization and `INITALE`

In principle, all that is needed to use `PINTofALE` is that the IDL system variable `!PATH` include the `.../PINTofALE/pro/` directory. As a matter of practical convenience however, it is advantageous to initialize some variables that are often used. This can be done using the script `initale.pro`, which defines various system variables (these are used by default in the various `.par` files, see above). For a list of variables defined and their hardcoded default values, see § B.91. All of these defaults may be overridden by redefining them in the file `PARFIL` (which by default is `.../PINTofALE/ardb/initale.par` but may be reset to any similarly structured file prior to running `initale.pro`). Note that these variables may also be redefined at any time at the IDL command line.

The initialization script `initale.pro` is a versatile routine that can be reused even in the middle of a session. If the variable `factory` is set to 1, it will reset all the variables to their original default values; if it `factory` is to 0, then only previously undefined variables will be defined.

On older versions of IDL, it turns out that a routine does not compile if a system variable that it uses has not been defined yet (even if it gets defined within the routine). In order to handle such cases, there is a "front-end" to `initale.pro` called `PoA_constructor.pro`. This can be run as a script, i.e.,

```
IDL> @PoA_constructor
```

which will in turn call `initale.pro` . It is expected that this script be run once only per session.

A point worth mentioning regarding the names of the defined system variables is that most of them have the same names as keyword names used in the programs. Notable exceptions are the names of the line and continuum emissivity directories `!LDBDIR` and `!CDBDIR` (the appropriate keyword name accepted by `rd_line()` and `rd_cont()` is `dbdir`) and the name of the file containing ion-balance data `!IONEQF` (corresponding to the keyword `eqfile` of `fold_ioneq()`). Other exceptions are `!GASPR`, `!LOGPR`, and `!EDENS`, corresponding to keywords `pres`, `logp`, and `n_e` respectively, of `rd_line()` and `rd_cont()`).

3.1.5 The ID Structure

A key concept of the use of `PINTofALE` for performing a line-by-line spectral analysis is the **ID Structure**. This is an IDL structure that includes line-specific information that can be manipulated.

Storing the line identification information in an unambiguous fashion is non-trivial because of problems such as the presence of line blends, multiple grating orders, instrument wavelength miscalibrations, etc., and hence all the relevant variables are carried together in a single data structure in order to facilitate bookkeeping. A feature in the spectrum is characterized by a wavelength, and has one set of identifications associated with it. Each ID consists of:

- One or more **wavelengths** corresponding to the identifications,
- **atomic numbers** of the atoms giving rise to the corresponding transitions,
- corresponding **ionic states**,
- the share of the observed **flux** in each of the components or the predicted flux if the former is unavailable,
- the **error** on the fluxes,
- the **line contribution function** for each line, including the ion balance information,
- the **temperatures** at which the above are defined, and
- a **description** of the transition, and any **comments** noted by the user.

Such a structure can easily account for the complex analysis requirements routinely encountered in spectral analysis. For example, a feature may have multiple identifications, of any element or ionic state, and the number of such identifications may be altered as needed. Identifications are stored by direct association and not by nearness to the wavelength of the observed feature, so that IDs from higher grating orders are correctly handled (the matched wavelengths are vastly different from the 1^{st}-order wavelength scale). For the same reason, wavelength shifts in the observed spectrum, which may arise either due to errors in wavelength calibration or due to doppler shifting of the source, also do not pose any problems.

Standardizing the output of the line identification process allows us to transparently handle the following steps such as flux measurement, derivation of the emission measure, deducing the plasma density, etc.

3.2 Interstellar Absorption

In the EUV and soft X-ray wavebands, even very nearby sources with relatively low ISM absorbing columns can be appreciably attenuated by H and He bound-free photoionizing transitions. It is also important to remember that neutral and once-ionized He can dominate the ISM absorption cross-section in the 44-200 Å (0.28-0.062 keV) range, and consideration of the neutral H absorption alone is generally not sufficient. Shortward of the C edge near 44 Å (0.28 keV), metals become the dominant absorbers.

The `PINTofALE` routine for computing the ISM absorbing optical depth is the function `ismtau()`, which includes different options for the absorption prescription in the soft X-ray range, and the ability to include the narrow auto-ionization resonance features in He I in the EUV (see, e.g., Rumph, Bowyer, & Vennes (1994) and references therein. In the soft X-ray range, three different absorption prescriptions are available, of which the first two are encoded in the current release: Morrison & McCammon (1983) polynomial fit between 1.2 and 400 Å (10-0.03 keV); cross-sections of Balucinska-Church & McCammon (1992); and Wilms, Allen, & McCray (2000) calculations that use cross-sections from Verner & Yakovlev (1995). The last two of these allow user-input abundances to be employed for the absorption computation (see the routine `getabund`). Absorption by molecular hydrogen is included according to the prescriptions of Cruddace et al. (1977), and Kashyap et al. (1992).

For illustration, the ISM transmittance ($e^{-\tau}$) computed using `ismtau()` for a "typical" mixture of neutral and ionized H and He H:He:He^{+} of 1:0.1:0.01 is illustrated in Figure 3.1 for the 5-200 Å range and for different values of the neutral hydrogen column.

The user should be aware that absorption edges in the soft X-ray range in all of the currently available prescriptions referenced above is still in the form of step functions and is very crude compared to the actual edge structure. Moreover, some older cross-sections, such as that of Morrison & McCammon (1983), included laboratory values for ionization edges that were measured from solid state samples rather than the gaseous atomic state. Consequently, there are errors in the edge energies that can be significant at the higher spectral resolutions of the *Chandra* and *XMM-Newton* grat-

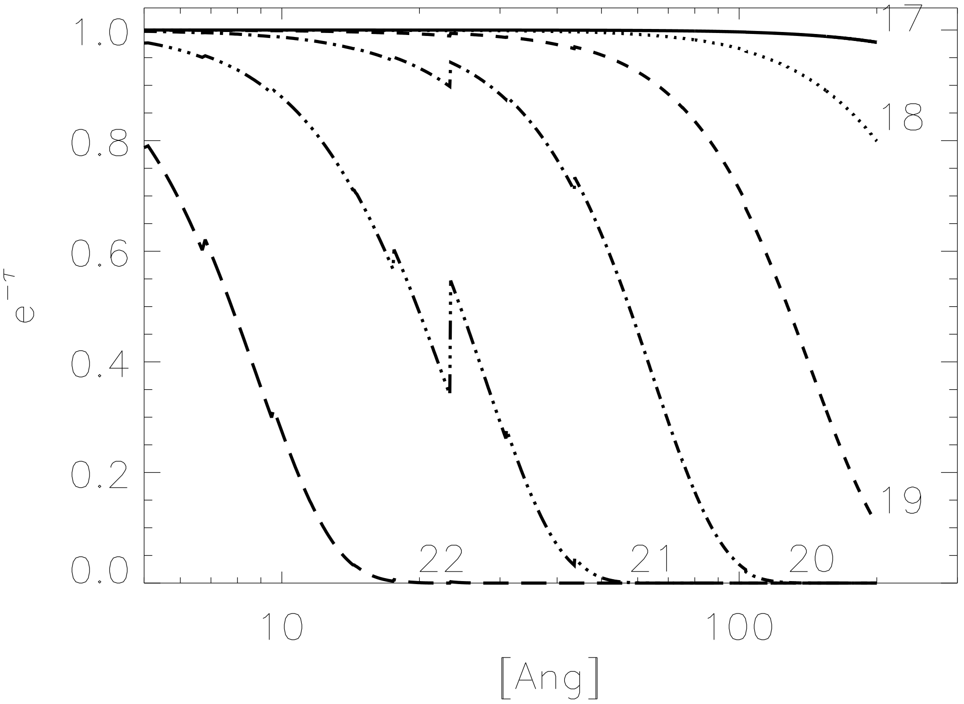

Figure 3.1: Illustration of the ISM transmittance in the 1-200 Å bandpass for different values of neutral hydrogen column density 10^{17}-10^{22} cm^2 computed using the `PINTofALE` routine `ismtau()`.

ing spectrographs. In the future, we will include available cross-sections with more accurate edge structure computed using, e.g., the R-MATRIX method employed recently by McLaughlin & Kirby (1998) for investigating the atomic O K edge region photoabsorption.

Note that there is an apparent inconsistency in the keywords for the neutral and once-ionized He columns in the calling of `ismtau()`. Ionized He is indicated by `HeII` (and lower/upper case combinations such as `hei`, `heii`, IDL being case insensitive), and neutral He by `He1` (and upper/lower case combinations). This arose because IDL forbids using `HeI` to denote neutral He, since it accepts abbreviated keywords and considers `HeI` to be the same as `HeII`.

3.3 Monte Carlo methods in `PINTofALE`

3.3.1 Monte Carlo Error Estimation

Error estimation via Monte Carlo is an efficient tool with which we determine the confidence bounds on model parameters in those cases where the fits involve prohibitively large numbers of parameters. Propagating errors for large parameter sets and handling it analytically or even numerically, especially when considering strong correlations that may exist between parameters (when covariance terms become important), may require an expert's knowledge of an array of statistical techniques that apply differently to different cases.

To illustrate, we use a simple example pertinent to our discussion of emission line spectroscopy. In Figure 3.2, we show a region of a *Chandra*/HETG spectrum of Capella where we have identified a strong resonance line from Ne IX and 3 contaminant lines originating from Fe XIX. A Gaussian profile is assigned to each of the four lines, each having 3 parameters: width, position, and flux. Together with a continuum level, they give a composite curve with which to fit the observed counts spectrum in this wavelength region. The Ne IX resonance line is part of a temperature-sensitive diagnostic, and measuring its flux accurately is generally of prime importance. Even after applying all known constraints, e.g., the widths of all the lines may be approximated by the instrument width, and line positions are quite well determined by referring to the atomic database, this situation presents a challenge to classical model fitting methods. Further, if we are to require reliable errors on the fit parameters, we must also account for the fact that because the profiles are so heavily blended with each other, their fluxes are strongly correlated. Thus, rather than proceeding with the task analytically,

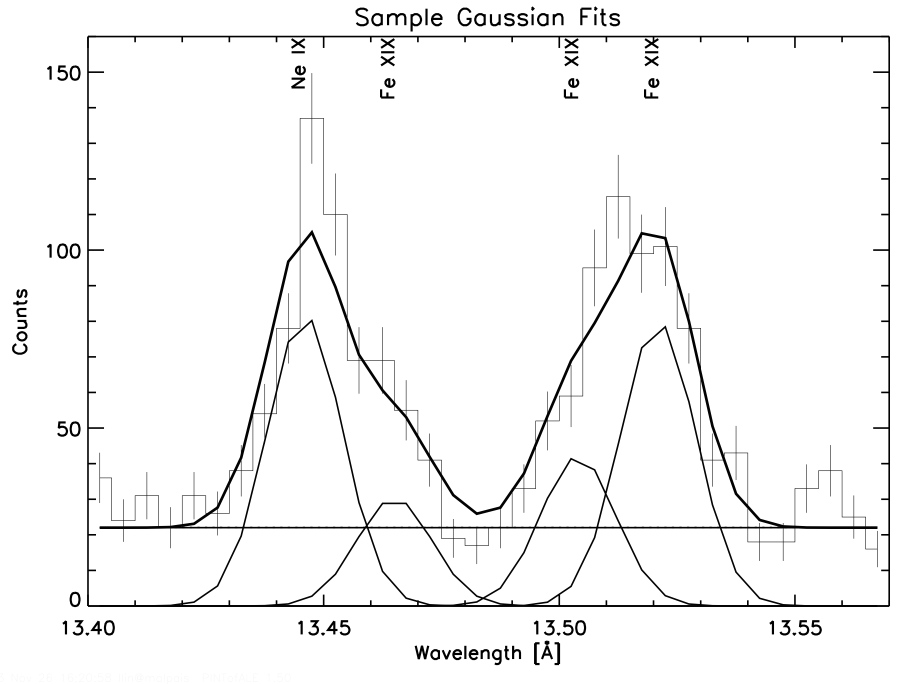

Figure 3.2: Sample fits of the heavily blended Ne IX region. Plotted over the spectrum (stepped histogram with error bars shown as vertical bars) are the continuum level (horizontal line), four Gaussian profiles for each line, and a composite of the five components. The fits were made using `PINTofALE` 's GUI-based line fitting software. Error analysis of such fits requires attention to strong correlations between the modeled integrated fluxes of the individual lines.

we take a Monte Carlo approach. Adhering to a method described in detail by Press et al. (2002), we:

1. create N synthetic data sets by varying the true data set randomly within its observed error
2. fit the N synthetic data sets in the same way that the original data set was analyzed, achieving N fitted sets of fluxes
3. characterize the uncertainty in our original fit by observing how the N sets of fluxes achieved synthetically are distributed about the original parameters. We may do this, e.g., by delineating confidence bounds

with a χ^2 statistic, or by measuring a confidence width in parameter space by counting simulated parameters starting from the true fit out towards a required confidence (e.g. 68% or 90%).

We have incorporated this method of error analysis in `PINTofALE`'s GUI based line fitting software. Preliminary tests of the method on single line fits (non-correlated parameters) give results which are largely in agreement with standard error computation methods.

3.3.2 Model Optimization with Markov Chain Monte Carlo

In general terms, the task of the astrophysicist often is to optimize a parameterized model to an observed data set to infer physically meaningful quantities with maximum confidence given the quality of the data. Given say, m data points $D = \{D_i, i = 1...m\}$ each with its associated error, and a proposed model with N parameters, $\Theta = \{\theta_\nu, \nu = 1...N\}$, the task is to find that Θ which best models the data according to some specified merit function. Put another way, we seek to maximize the probability of observing parameters D given a model Θ, by assessing for example $P(D|\Theta) = exp(-1/2\sum_{i=1}^{m}((D_{i_{obs}} - D_{i_{mod}})/\sigma_{i_{obs}})^2)$ for different sets of Θ, with $D_{i_{mod}}$ being the data set calculated using the model evaluated for Θ, to compare with an observed data set $D_{i_{obs}}$ with associated error $\sigma_{i_{obs}}$.

It is often the case that one may proceed fruitfully by employing standard methods such as Levenberg-Marquardt minimization, which calculates gradients in parameter space to use as a guide in an iterative stepping process towards areas of high likelihood, after which a solution to the N dimensional problem is approximated. Such a process can however become ill-suited if one encounters a problem with a relatively large number of parameters characterizing the model for a middling number of data points.

In an ideal world, where computational power is no object, one can envision that such complex parameterized problems can be solved by a brute force Monte Carlo process. A copious amount of sets Θ can be randomly generated and could sample a merit function in N dimensional parameter space with resolution adequate to not only give a good estimate of the best probability set Θ but also the projected confidence or credible bounds in the ascertained parameters. Realistically, a Monte Carlo approach would require some sort of conditioning mechanism to make those few simulations which are computationally tractable to actually count. A Markov Chain is exactly such a mechanism. A Markov chain is an ordered sequence of random variables (in our case parameter sets), $\Theta_0, \Theta_1, \Theta 2, ...$, in which the likelihood that a certain set Θ_{i+1} appear in the sequence is directly dependent only on Θ_i. Put in terms of conditional probabilities: $P(\Theta_{j+1}|\Theta_j, ...\Theta_0) = P(\Theta_{j+1}|\Theta_i)$. A set of transition rules can be adopted to govern the evolution of variables within such a sequence, For example, we adopt two such transition rules: First, the candidates for the next parameter set $\Theta^\star$ in sequence are obtained as Normal deviates of the current parameter set Θ: $\theta^\star = \theta + r\sigma$ where r is a random number and σ is an estimate of the variance of θ; and second, the Metropolis criterion (Metropolis 1953), where candidate parameter sets are adopted with a probability $A(\Theta, \Theta^\star) = min[1, \frac{P(D|\Theta^\star)}{P(D|\Theta)}]$, i.e. a new parameter set is sometimes adopted even if its associated probability is smaller than that of the previously adopted set, a tactic designed to avoid having the solution get bogged down in local minima.

The discerning nature of the Markov Chain coupled with the randomized search of the Monte Carlo provides a computationally feasible method to harness the power of Monte Carlo. The 'random walk' nature of the MCMC approach allows the algorithm to probe regions of parameter space which might otherwise be ignored (e.g., χ^2 minimization routines often follow the steepest gradients in parameter space, not necessarily the gradient leading to the deepest valley). Moreover, simulations are run in batches, assuring adequate sampling of local parameter space, and would make apparent any non-uniqueness in the parameter solution. For a review of MCMC methods see Neal (1993).

A generalized fitting engine employing MCMC is in the process of being built and incorporated into `PINTofALE`. In the following, we describe one instance of such an algorithm, one developed specifically to address the issue of stellar coronal DEMs.

3.3.3 Markov-Chain Monte Carlo DEM Reconstruction

In the language used to describe MCMC above, the DEM reconstruction problem is to maximize the probability of a modeled DEM and abundances given a set of measured line or continuum fluxes or flux ratios. That is, the task is to assess a merit function $P(D|\Theta) = exp(-1/2\sum_{i=1}^{m}((I_{i_{obs}} - I_{i_{mod}})/\sigma_{i_{obs}})^2)$ where Θ is a parameter set used to model the DEM, $I_{i_{obs}}$ are measured fluxes with associated error $\sigma_{i_{obs}}$, and $I_{i_{mod}}$ are model fluxes calculated using a DEM model evaluated for Θ and using emissivity curves from an atomic database. [1] Note that the exponent is just the well known

---

[1] It is possible to use other statistics, such as Cash or Castor (Cash 1979) to determine the likelihoods in the Poisson regime. However, the $e^{-\frac{1}{2}\chi^2}$ statistic used here by default is the best choice for DEM analysis, since the fluxes input for analysis are usually in intensity

expression for χ^2.

There are several approaches to how one can parameterize the DEM. These can be grouped into two general categories: continuous models which parameterize using a few control parameters (e.g. splines or Chebychev polynomials) and discrete models, where the DEM is an array of N temperature bins, giving essentially N model parameters (e.g. Singular Value Decomposition, the Regularization Method, genetic algorithms). Continuous models truncate the number of free parameters, thereby making the problem tractable for common minimization techniques (e.g. Levenberg-Marquardt). Their major drawback however is that assuming a functional form can impose non-physical constraints on the shape of the DEM. Also, probing a parameter space with standard χ^2 minimization techniques is prone to find local minima and will sometimes incur parameter degeneracy and non-unique solutions. The major advantage of a discretized parameterization is that it can approximate both continuous and discontinuous functional forms for the DEM. Its ability to do so however, depends on how fine one is willing to define the binning scheme and what constraints one places on the smoothness of the final solution. One can imagine for example, a DEM defined between $Log(T) = 5.5$ and $Log(T) = 7.8$ discretized with modest 0.1 *dex* bins will give 23 free parameters. Such a parameterization is intractable with e.g., Levenberg-Marquardt, as the number of data points available is generally limited to the $\simeq 10$ strong emission lines and/or flux ratios available in the spectra.

Kashyap & Drake (1998) offer the latter approach to DEM reconstruction problem using a Markov-Chain Monte Carlo process. Their approach is now made available via the `PINTofALE` procedure `MCMC_DEM`. The procedure is highly flexible, offering the user a range of inputs allowing users to control the number of simulations, specify line ratios, allow different types of smoothing, censor data points (i.e., use upper and lower limits), etc. The output of the simulations are stored and are an ideal resource to characterize the uncertainty in the reconstructions.

It is important to realize that the MCMC algorithm implemented here cannot be used as a black box. While highly flexible and capable of addressing almost any type of astronomical dataset, it does require of the user a measure of understanding of the process. This is the first algorithm of its kind made generally available to the astronomical community: there is no other DEM reconstruction method that computes uncertainties as a matter of course, nor one so general, and no MCMC based reconstruction algorithm.

---

units (not photon counts) and are also background subtracted.

The complexity of the algorithm can be seen in the data flow diagram setting up the *input* to `MCMC_DEM` (Figure 3.3) and the data flow diagram of the MCMC DEM reconstruction algorithm itself (Figure 3.4). This process is of course also subject to yet more systematic uncertainties that arise from an imperfect knowledge of the underlying atomic data and the approximations made to the physics of the source (see Judge et al. 1997; Kashyap & Drake 1998).

Application and Summary

We have tested the `PINTofALE` procedure `MCMC_DEM` exhaustively, both on synthetic spectra as well as actual high-resolution spectral data. One such test is shown as an example here. We first compute a synthetic spectrum (assuming Gaussian errors of 10 %) after adopting an arbitrary DEM with a double Lorentzian structure (maxima at $10^{6.6}$ K and 10^{7}.1 K; see Figure 3.5). The spectrum is generated assuming solar abundance values (Grevesse & Sauval 1999). From this spectrum, a set of lines from H-like and He-like species were identified (using `LINEID()`). Lines from some of the most abundant species, O, Ne, Mg, Si, and S were chosen because they are relatively strong and their atomic data are the most reliable and well understood. Using `PINTofALE`, we identified a set of 24 of these lines and measured line fluxes by fitting Gaussian profiles (`FITLINES()`) and correcting for the instrument effective area. We then set up the MCMC analysis (see Figure 3.3) by reading in the appropriate emissivities.

We ran an abundance invariant DEM reconstruction analysis using H-like/He-like ratios of lines from common elements (e.g. O VII/O VII, Ne IX/Ne X, Fe VIII/Fe VII etc.)(Drake et. al. 2004). A smoothing scale derived internally based on the widths of the emissivity functions was used to damp out the high-frequency oscillations inherent in this type of integrals ; this is a Fredholm integral of the 1^{st} kind). In a preliminary analysis, we run 500 simulations, which required around 7 minutes with an AMD Athlon 1800 Mhz processor.

The result of the reconstruction are shown as the shaded band in Figure 3.5, where the width of the band tracks the uncertainty in the DEM. All the simulated DEMs within the 1-σ level are shown. We can also characterize the uncertainty by plotting the 50% most probable DEMs (Figure 3.6). Both Figure 3.5 and Figure 3.6 were made with `MCMC_PLOT` an accessory routine to `MCMC_DEM`).

This MCMC based DEM reconstruction technique has also been applied to the analysis of many X-ray active stars: Drake et al. (2001) have applied it to ξ Boo A, Maggio et al.(2002) to solar analogs AD Leo and π^1 UMa, Argiroffi et al. (2003) to Capella (α Tau), Ball et al. (2004) to ξ UMa, and Garcia et al. (2004) to AB Dor and V471 Tau, all using beta versions of `PINTofALE` MCMC software.[2]

---

[2] A guide to using `MCMC_DEM` is available online at `http://hea-www.harvard.edu/PINTofALE/doc/MCMC_DEM.html`.

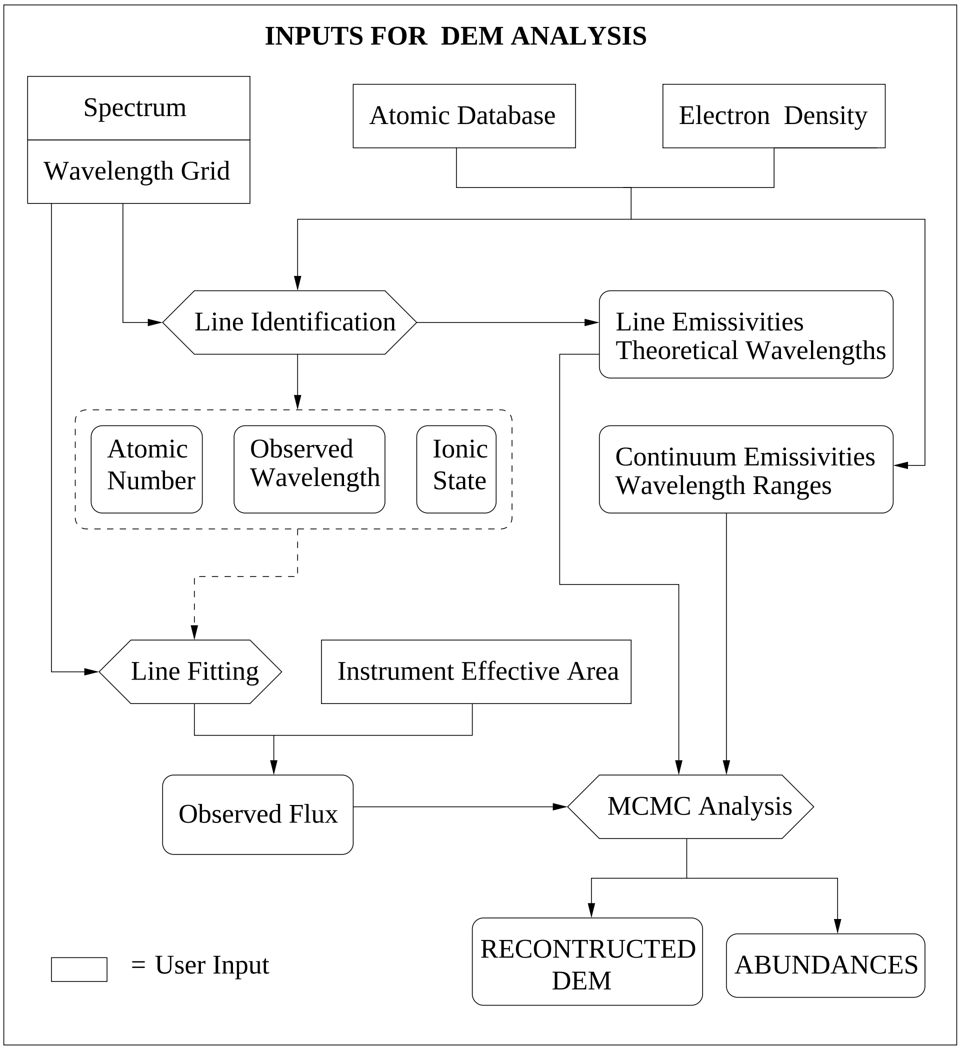

Figure 3.3: Data flow chart of input parameters required for MCMC_DEM(). The direction of the arrows correspond to the direction of the data flow. Square rectangles indicate user inputs. Hexagons indicate analysis using `PINTofALE` software. Rounded rectangles indicate substantial analysis operations which have been generally encoded as stand-alone routines in `PINTofALE` . Rounded rectangles indicate analysis products.

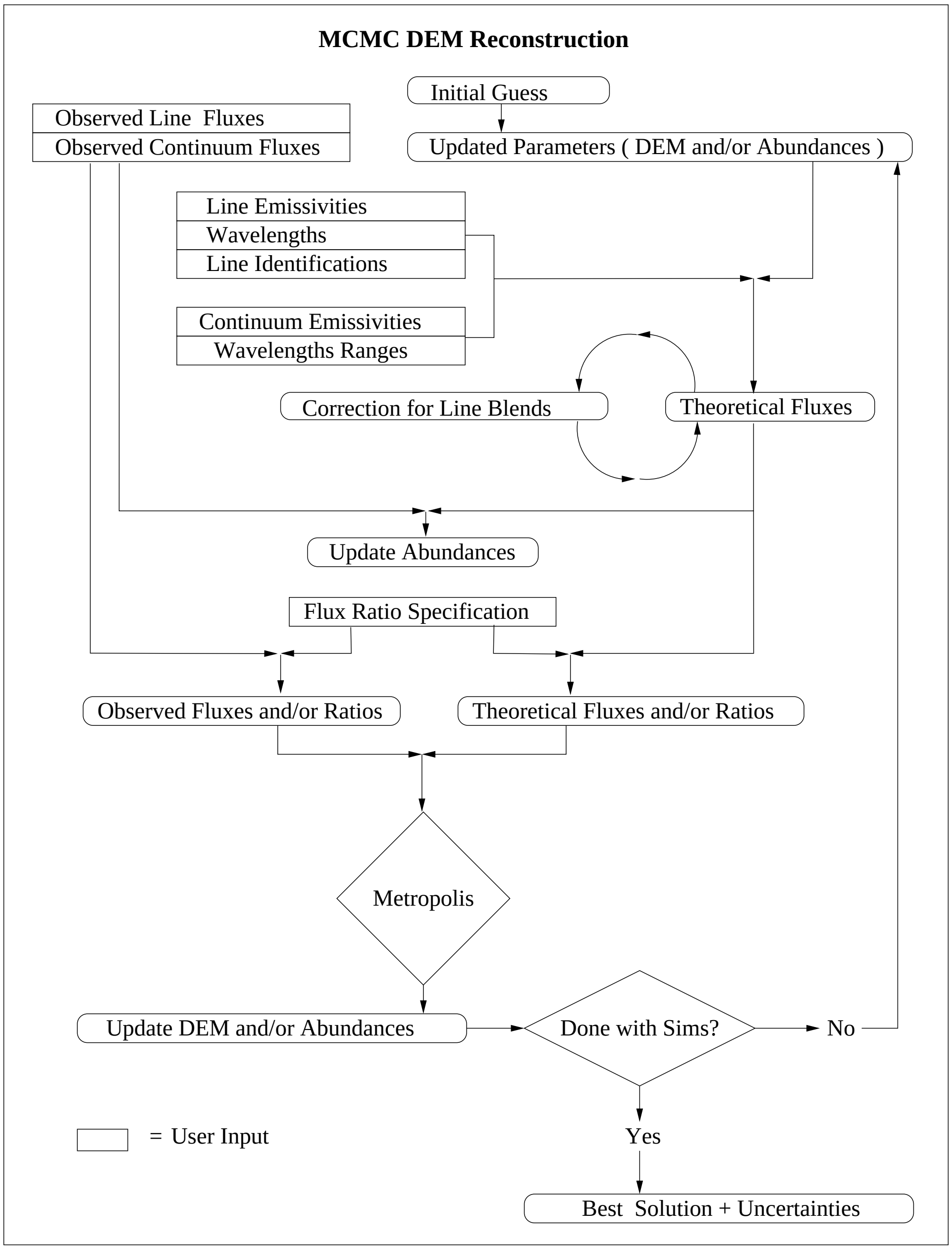

Figure 3.4: Data flow chart depicting the DEM realization process within MCMC_DEM(). The direction of the arrows indicate the direction of the data flow. Square rectangles indicate user inputs. Rounded rectangles indicate a computational result. Diamonds indicate a logic gate.

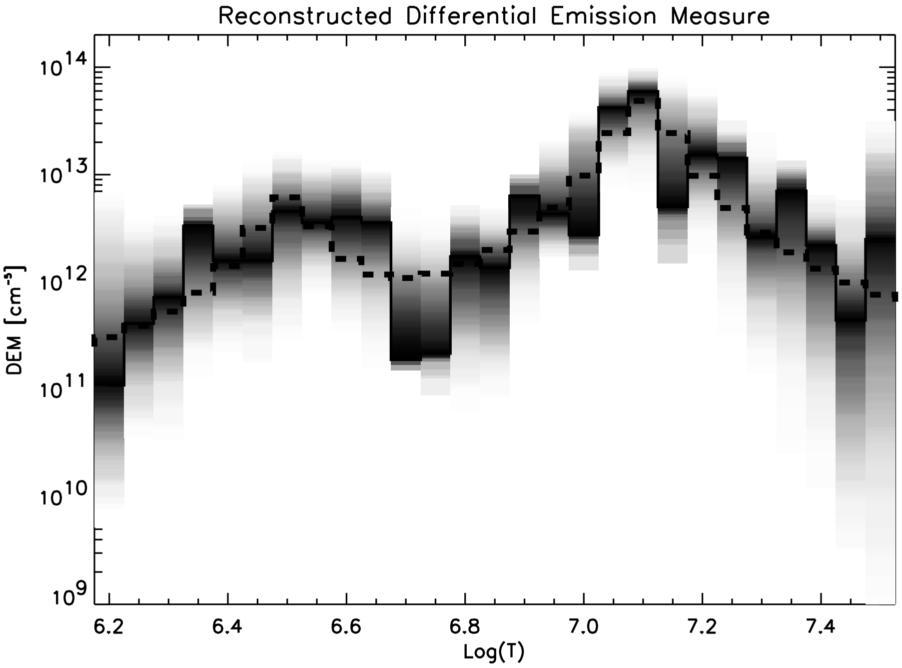

Figure 3.5: The reconstructed DEM distribution compared with the original. The original is plotted with a thick, dashed black line. In the top panel, the uncertainty in the DEM is displayed by plotting simulated emission measures within one sigma of the best value for each bin.

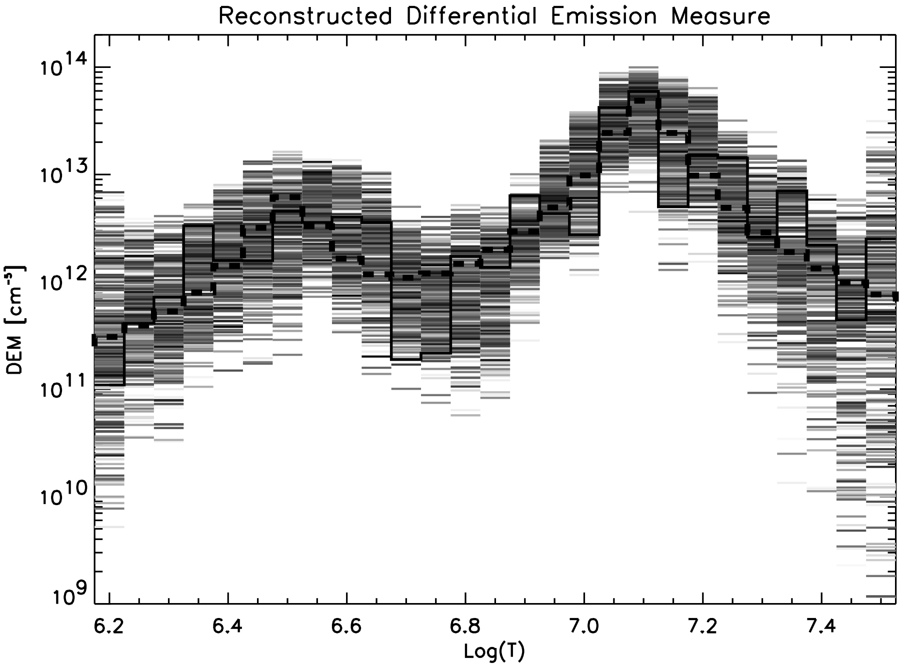

Figure 3.6: The reconstructed DEM distribution compared with the original. The uncertainty in the DEM is displayed by plotting all DEM realizations with the shade of grey indicating the probability associated with each. Darker shades indicate a better probability.

Chapter 4

Using `PINTofALE`

The following assumes that a working version of `PINTofALE` has been installed, together with the formatted emissivity databases. Information on how to generate the CHIANTI database starting from a standard CHIANTI release was presented earlier in 2.

There are two main example sessions included here that run through some of the features of `PINTofALE`. The first is an "explicit" session in which procedures and functions are called on the command line with all the appropriate arguments and keywords defined. This is the method that requires the most typing, and the most clear understanding of the `PINTofALE` procedures and functions being used.

The second is an example session using `.par` files that more or less follows the same operations that the explicit session performs. The `.par` files contain within them the call to a function or procedure, together with all the arguments and keywords, with as many of these set to standard variable and system variable names as is convenient. The `.par` method is the one that requires the least typing, and the least understanding of the innards of `PINTofALE` and of the exact meanings and functions of all the arguments and keywords. Convenience comes at a price however: the `.par` method can be more dangerous if attention is not paid to what is being done within the `.par` files.

4.1 Threads and Example Sessions

4.1.1 Explicit Session

```
A detailed example session
```

```
   This annotated example walks a user through a PINTofALE session,
   sparing no details however gory, unlike the similar thread that uses
   .par files. During this run, the user will
    1. read in the line and continuum emissivity databases,
    2. compute a theoretical spectrum,
    3. browse spectra with special attention to lines of specific species,
       density sensitive lines, and unblended lines, etc.
    4. identify line features in the observed spectrum,
    5. derive line fluxes by fitting line profiles,
    6. compute emission measures appropriate for the derived fluxes, and
    7. create a LaTeX table of the identifications
     ------------------------------------------------------------------

0. INITIALIZE THE SETTINGS AND LOAD THE DATA

#       start IDL
idl

;       set up the PINTofALE environment
;
.run initale

;NOTE:
;       if INITALE fails, run the script
;       @PoA_constructor

;       load the data and instrument files
;
restore,!ARDB+'/demo_v711Tau.save',/verbose

;       copy the relevant variables to more normal looking names
;
data_spec     = SPC_M1P_V711TAU ;observed spectrum [ct/bin]
data_wvl      = LAM_M1P_V711TAU ;wavlengths for observed spectrum [Ang]
data_spec_err = SPE_M1P_V711TAU ;errors on observed spectrum [ct/bin]
effar         = EFFAR_M1P       ;effective area [cm^2]
wvlar         = WVLAR_M1P       ;wavelengths for effective area [Ang]

;       and plot them
;
;               counts spectrum and errors
;
plot, data_wvl,data_spec,psym=10,xtitle='Wavelength ['+!AA+']',$
  ytitle='Counts/bin',title='HR 1099: HETG/MEG +1',xrange=[5.,20.],$
  yr=[0.1,1200],/ys
```

```
for i=0L,n_elements(data_wvl)-1L do oplot,$
  data_wvl[i]*[1,1],data_spec[i]+data_spec_err[i]*[-1,1],$
  color=1 & setkolor,'red',1
;
;               instrument effective area
;
plot, wvlar,effar,xtitle='Wavelength ['+!AA+']',$
  ytitle='Effective Area [cm!u2!n]',title='HETG/MEG +1',xr=[5.,20.]
;
;               normalized spectrum
;
tmp_effar=interpol(effar,wvlar,data_wvl)
plot, data_wvl,data_spec/tmp_effar/EXPTIM_V711TAU,$
  psym=10,xtitle='Wavelength ['+!AA+']',ytitle='Counts [cm!u-2!n s!u-1!n]',$
  title='HR 1099: HETG/MEG +1',xr=[5.,20.]
;
;               DEM
;
plot, LOGT_V711TAU,DEM_V711TAU,psym=10,/ylog,xtitle='log!d10!n (T [K])',$
  ytitle='DEM [cm!u-5!n]',title='EUVE based approximate DEM for HR1099'

     ------------------------------------------------------------------

1. READ IN THE LINE AND CONTINUUM DATABASE

;       set the wavelength range of interest
;
!WMIN = min(data_wvl)
!WMAX = max(data_wvl)

;       read in the line emissivity database
;       [1e-23 ergs cm^3 s^-1]
;
rd_line_linemis = rd_line(atom, n_e=!EDENS, wrange=[!WMIN,!WMAX], $
  dbdir=!LDBDIR, wvl=rd_line_wvl, logT=rd_line_logT, z=rd_line_z, $
  ion=rd_line_ion, jon=rd_line_jon, src=rd_line_src, fstr=rd_line_linstr, $
  verbose=!VERBOSE)

;       fold in line emissivities with ion-balance info
;       (this is needed only if the input emissivities do not already
;       contain the ion-balance info.  For example, this is needed while
;       using our implementation of the SPEX and CHIANTI line emissivity
;       databases.  on the other hand, APED, as well as the continuum
;       emissivity databases, come with ion-balance already folded in.)
;
```

```
rd_line_linemis = fold_ioneq(rd_line_linemis, rd_line_Z, rd_line_JON, $
  logT=rd_line_LOGT, tmax=fold_ioneq_tmax, trng=fold_ioneq_trng, level=0.1, $
  chifil=chifil, chidir=!CHIDIR, eqfile=!IONEQF, verbose=!VERBOSE)

;NOTE 1:
;       it may be advantageous (and less of a headache later) to overwrite
;       the corrected emissivities into the line emissivity structure.
;       the main output, rd_line_linemis, will still have emissivites
;       without ion balance folded in.
;
rd_line_linstr.LINE_INT = rd_line_linemis

;NOTE 2:
;       the line emissivities are read in from an existing database
;       computed for a density nearest to that specified.  In order
;       to interpolate from emissivities computed at two densities,
;       it is necessary to read in each set and to manually interpolate.
;       here is one way that it can be accomplished.
;
;       In practice, this should not be necessary unless synthesis of very
;       density-sensitive features is required at a very specific density.
;       Remember that this interpolation is only linear in log.
;
n_e=5e10 & n_e1=1e10 & n_e2=1e11
lne=alog10(n_e) & lne1=alog10(n_e1) & lne2=alog10(n_e2)
weight1=(lne-lne1)/(lne2-lne1) & weight2=(lne2-lne)/(lne2-lne1)
;
rd_line_linemis1=rd_line(atom,n_e=n_e1,wrange=[!WMIN,!WMAX],dbdir=!LDBDIR,$
  wvl=rd_line_wvl,logT=rd_line_logT,z=rd_line_z,ion=rd_line_ion,$
  jon=rd_line_jon,src=rd_line_src,verbose=!VERBOSE)
;
rd_line_linemis2=rd_line(atom,n_e=n_e2,wrange=[!WMIN,!WMAX],dbdir=!LDBDIR,$
  wvl=rd_line_wvl,logT=rd_line_logT,z=rd_line_z,ion=rd_line_ion,$
  jon=rd_line_jon,src=rd_line_src,verbose=!VERBOSE)
;
rd_line_linemis=10.^(weight1*alog10(rd_line_linemis1>1e-30)+$
  weight2*alog10(rd_line_linemis2>1e-30))
;
;       and do not forget to include ion balance again..
;
rd_line_linemis=fold_ioneq(rd_line_linemis,rd_line_Z,rd_line_ION,$
  logT=rd_line_LOGT,tmax=fold_ioneq_tmax,trng=fold_ioneq_trng,level=0.1,$
  chifil=chifil,chidir=!CHIDIR,eqfile=!IONEQF,verbose=!VERBOSE)
;
;       .. and push the emissivities into the structure
```

```
;
rd_line_linstr.LINE_INT = rd_line_linemis

;       read in the continuum emissivity database
;       [1e-23 ergs cm^3 s^-1 Ang^-1]
;       (the continuum emissivities are very weakly dependent on
;       changes in ion balance, and so it is already included in
;       the databases)
;
rd_cont_conemis = rd_cont(!CEROOT, n_e=!EDENS, wrange=[!WMIN,!WMAX], $
  dbdir=!CDBDIR, abund=!ABUND, metal=1., wvl=rd_cont_cwgrid, $
  logT=rd_cont_LOGT, fcstr=rd_cont_constr, verbose=!VERBOSE)
```

---

2. THEORETICAL SPECTRUM

There are many different programs in PINTofALE (e.g., LINESPEC, LINESPEC_EM, MAKE_SPECTRUM, etc.) that will allow one to generate a theoretical spectrum. The basic scheme however is to first compute observed line fluxes from the line emissivities and bin them into a spectrum, and add the continuum spectrum, derived by computing the fluxes and rebinning to the same wavelength grid as the line spectrum. Below we will illustrate this scheme in detail, first making a complete spectrum, and second exploring smaller parts of the spectrum. Note that this section requires variables defined in the previous section.

1. Compute Spectrum
2. Subsets of Spectrum

2.1 COMPUTE SPECTRUM

```
;       first make sure that the DEM is defined on the same temperature
;       grid as are the emissivities
;
!DEM = mk_DEM('interpolate', logT=!LOGT, indem=DEM_V711TAU, $
  pardem=LOGT_V711TAU)

;NOTE:
;       we require that the DEM always be defined in the form
;               n^2*dV/dlnT
;       modifications such as n^2 dh/dlnT, or division by (4 pi d^2),
;       or multiplication by (R^2/d^2) are OK too -- as long as
;       the user keeps track of things.  For the sake of definiteness,
;       in what follows, we will assume that the units are [cm^-5]
```

```
;       compute the line fluxes
;       emissivities are [ergs cm^3/s], DEM is [/cm^5], effective
;       area is [cm^2].  the [ergs] are converted internally to [ph],
;       so the output of LINEFLX has the units [ct/s]
;
lineflx_flx = lineflx(rd_line_linemis, rd_line_LOGT, rd_line_WVL, rd_line_Z, $
  DEM=!DEM, abund=!ABUND, effar=effar, wvlar=wvlar) * EXPTIM_V711TAU

;       units at this stage: [ct]

;       correct for ISM absorption
;
lineflx_flx = lineflx_flx*$
  exp(-ismtau(abs(rd_line_WVL), NH=!NH, fH2=!FH2, He1=!HE1, HeII=!HEII, $
  /bam, abund=!ABUND, verbose=!VERBOSE))

;       define a wavelength grid to hold the spectrum
;
sp_NBIN  = 20000L                               ;number of bins
sp_DW    = (!WMAX-!WMIN)/sp_NBIN                ;bin width
sp_wgrid = findgen(sp_NBIN+1L)*sp_DW+!WMIN      ;bin boundaries

;       bin the line fluxes into a spectrum
;
linspec = hastogram(abs(rd_line_WVL), sp_wgrid, wts=lineflx_flx)

;       compute continuum fluxes
;       emissivities are [ergs cm^3/s/A], DEM is [/cm^5], effective
;       area is [cm^2].  the [ergs] are converted internally to [ph],
;       so the output of LINEFLX has the units [ct/s/A], and is
;       multiplied by the bin widths [A] and the exposure time [s].
;
rd_cont_CDW = rd_cont_cwgrid[1:*]-rd_cont_cwgrid
;
contflx_flx = lineflx(rd_cont_conemis, rd_cont_LOGT, rd_cont_constr.MIDWVL, $
  DEM=!DEM, effar=effar, wvlar=wvlar) * rd_cont_CDW * EXPTIM_V711TAU

;       units at this stage: [ct]

;NOTE:
;       LINEFLX, despite its name, is a fairly general routine, which
;       integrates the emissivity function over the temperature grid
;       weighted by the DEM at each wavelength.  The fact that we might
;       be dealing with continuum emissivities rather than line emissivities
;       is immaterial to the program.  Also, when the atomic numbers are
```

```
;       omitted, all the wavelengths are assumed to belong to H, with
;       an abundance of 1.0, and hence the output will be perfectly fine.

;       correct for ISM absorption
;
contflx_flx = contflx_flx*$
  exp(-ismtau(rd_cont_constr.MIDWVL, NH=!NH, fH2=!FH2, He1=!HE1, $
  HeII=!HEII, /bam, abund=!ABUND, verbose=!VERBOSE))

;       rebin
;
conspec = rebinw(contflx_flx, rd_cont_cwgrid, sp_wgrid, /perbin)

;       add the line and continuum spectra
;
spec = linspec + conspec

;       plot the result
;
plot, sp_wgrid,spec,psym=10,xr=[5.,30.],/xs,yr=[1e-2,5e3],/ys,/ylog,$
  xtitle='Wavelength ['+!AA+']',ytitle='Counts [ct]',$
  title='Theoretical spectrum'
oplot, sp_wgrid,linspec,color=1,psym=10
oplot, sp_wgrid,conspec,color=2,psym=10
setkolor, 'red',1 & setkolor, 'yellow',2

;       convolve with lorentzian of half-width=5*bin-width to simulate LRF
;       This is an example of a "general utility" PINTofALE routine that
;       is not coronal plasma specific.  Use a "Lorentzian beta model" with
;       exponent 1.8 .
;
lrfspec = voorsmooth(spec, 5., type='beta=1.8')

;       plot the result
;
plot, sp_wgrid,spec,psym=10,xr=[5,8],/xs,/ylog,$
  xtitle='Wavelength ['+!AA+']',ytitle='Counts [ct s!u-1!n]',$
  title='Predicted spectrum'
oplot, sp_wgrid,lrfspec,color=1,psym=10,thick=2

;NOTE:
;       these variables have been saved in !ARDB/example_1.save thusly:
;save,file=!ARDB+'/example_1.save',sp_wgrid,spec,lrfspec,linspec,conspec

;       restore stored values to continue with current demo
```

```
;
restore,!ARDB+'/example_1.save',/verbose

2.2 SUBSETS OF SPECTRUM

   The above will give a "complete" spectrum with all the lines in the
   database. To compute, e.g., just the spectrum of Ne with no continuum
   is easy by selecting only the lines from elements with atomic number =
   10
;       select the lines of interest
;
ae=where(rd_line_Z eq 10)

;       make the spectrum (note that using the full wavelength range
;       of sp_wgrid is a bit inefficient here though)
;
linspec_Ne = hastogram(abs(rd_line_WVL[ae]), sp_wgrid, wts=lineflx_flx[ae])

;       convolve with the "LRF"
;
lrfspec_Ne = voorsmooth(linspec_Ne, 5., type='beta=1.8')

;       plot the result
;
!p.multi=[0,1,2]
plot, sp_wgrid,linspec_Ne,psym=10,xr=[9,14],/xs,$
  xtitle='Wavelength ['+!AA+']',ytitle='Counts [ct]',$
  title='Predicted Ne spectrum'
oplot, sp_wgrid,lrfspec_Ne,color=1,psym=10,thick=2
;       zoom in on Ne IX triplet
plot, sp_wgrid,linspec_Ne,psym=10,xr=[13.3,13.8],/xs,$
  xtitle='Wavelength ['+!AA+']',ytitle='Counts [ct]',$
  title='Ne IX triplet'
oplot, sp_wgrid,lrfspec_Ne,color=1,psym=10,thick=2
!p.multi=0
     ------------------------------------------------------------------

3. SPECTRAL BROWSING

   This section illustrates some interactive ways to locate
   ``interesting'' regions and features in an observed spectrum by linking
   it to the database read in above via theoretically biased filters.
    1. Locate Specific Lines
    2. Explore Strong Lines
    3. Strong Lines of Particular Species
```

```
    4. Density sensitive Lines
    5. Unblended Lines
    6. Instrument Features

3.1 LOCATE SPECIFIC LINES

;       find locations of lines of a given element and ionisation stage,
;       say, all lines of Ne.

ae=where(rd_line_linstr.z eq 10)
show_line, abs(rd_line_wvl[ae]), Z=rd_line_Z[ae], ion=rd_line_ion[ae], $
  lambda=data_wvl, spec=data_spec, order='1,3', sep=',', xo=xo, yo=yo, $
  oxr=oxr, oyr=oyr, title='Looking for Neon'

;NOTE:
;       SHOW_LINE calls the "general utility" routine PICKRANGE,
;       which is described elsewhere.  To test out features of
;       PICKRANGE, try zooming in on a rectangular region drawn with
;       the click-and-drag of the right mouse button, followed by
;       left button click to enact the zoom; when an interesting
;       feature has been found, get cursor control with a left button
;       click, then dump a postscript hardcopy by typing h (can
;       toggle color, encapsulated postscript, or full landscape
;       by typing c, e, and l), then = to dump the file.


3.2 EXPLORE STRONG LINES

;       To get a rough first idea of line ID's for prominent lines,
;       just select lines that are stronger than 1/10 the flux of
;       the strongest line

ae=where(lineflx_flx/max(lineflx_flx) gt 0.1)
show_line, abs(rd_line_wvl[ae]), Z=rd_line_Z[ae], ion=rd_line_ion[ae], $
  lambda=data_wvl, spec=data_spec, order='1,2,3', sep=',', xo=xo, yo=yo, $
  oxr=oxr, oyr=oyr, title='Expected strong lines'

;NOTE:
;       continuing to test PICKRANGE: try, e.g., browsing through
;       the spectrum by zooming in on a narrow wavelength range to
;       the short wavelength end (right button click and drag, then
;       click with left button to select), then click again with
;       left button to get keyboard control; now > and < scroll
;       through the spectrum to right and left.
```

```
3.3 STRONG LINES OF PARTICULAR SPECIES

;       One more: Fe XVII lines, but only the stronger ones again,
;       and scaling the y position of the labels according to the
;       theoretical fluxes from lineflx, using ~200 as peak of
;       Fe XVII resonance line at 15.015 AA, and adding to very
;       rough continuum derived above (section 2).

aei=where(rd_line_Z eq 26 and rd_line_ION eq 17)
ae=where(rd_line_Z eq 26 and rd_line_ION eq 17 and $
  lineflx_flx/max(lineflx_flx[aei]) gt 0.05)
in_x=abs(rd_line_wvl[ae])
in_y=200.*lineflx_flx[ae]/max(lineflx_flx[ae]) + $
  interpol(conspec,0.5*(sp_wgrid[1:*]+sp_wgrid),in_x)
;
show_line, in_x, in_y, Z=rd_line_Z[ae], ion=rd_line_ion[ae], $
  lambda=data_wvl, spec=data_spec, order=’1,2,3’, sep=’,’, $
  xo=xo, yo=yo, oxr=oxr, oyr=oyr, title=’Fe XVII’

;       Note that zooming in here does not change y label
;       location because y positions are fixed by in_y

3.4 DENSITY SENSITIVE LINES

;       find density sensitive lines using LSD.  This routine needs
;       to read in the emissivities for different densities in order
;       to find which lines have changed flux.  Set the threshold for
;       density sensitivity to factor of 1.5

lsd_list = lsd([!WMIN,!WMAX], lsd_flx, lsd_wvl, edens=[1.e9,1.e13], $
  DEM=!DEM, tlog=!LOGT, dbdir=’$SPEX’, ceiling=5., flor=1.e-2, $
  ratmax=lsd_ratmax, flxmax=lsd_flxmax, outZ=lsd_Z, outIon=lsd_Ion, $
  chifil=1, chidir=!CHIDIR, eqfile=!IONEQF, abund=!ABUND, $
  effar=effar, wvlar=wvlar)
show_line, lsd_wvl, Z=lsd_Z, ion=lsd_ION, lambda=data_wvl, spec=data_spec, $
  order=’1,2,3’, sep=’,’, xo=xo, yo=yo, oxr=oxr, oyr=oyr, $
  title=’Density Sensitive Lines’

3.5 UNBLENDED LINES

;       find lines that are relatively unblended, say less than 20%.

;       First set the "resolution" - the +/- delta wavelength from
;       line centre within which to consider neighbouring lines as
;       blended.
```

```
;
in_dwvl=0.015           ;in [AA], reasonable rough number for MEG

;       figure out the blending factor for all the lines
;
bland_frac = bland(rd_line_linstr, in_dwvl, flx=lineflx_flx, $
  wflx=abs(rd_line_wvl))

;       choose the "least blended"
;
ae=where(bland_frac gt 0.8 and lineflx_flx/max(lineflx_flx) gt 0.1)

;       and display them
show_line, abs(rd_line_wvl[ae]), Z=rd_line_Z[ae], ion=rd_line_ION[ae], $
  lambda=data_wvl, spec=data_spec, order='1,2,3', sep=',', xo=xo, yo=yo, $
  oxr=oxr, oyr=oyr, title='Unblended Lines'

3.6 INSTRUMENT FEATURES

;       Now run SMUDGE to find instrument features and use in
;       SHOW_LINE too, first selecting the instrument configuration
;       (this is accomplished by logical contructions using keywords
;       keys (OR), only (AND) and nisi (NOT) in SMUDGE - yes, it is
;       not so obvious but was done that way to avoid a huge and
;       repetitive instrument information database file
;
smudge, smudge_loc, smudge_label, keys='HRMA HEG', range=[!WMIN,!WMAX], $
  infile=!ARDB+'/smudge.lst'

;       and display them _along with_ the unblended lines
show_line, abs(rd_line_wvl[ae]), Z=rd_line_Z[ae], ion=rd_line_ION[ae], $
  lambda=data_wvl, spec=data_spec, wmark=smudge_loc, lmark=smudge_label, $
  order='1,2,3', sep=',', cmark=150, markc=222, xo=xo, yo=yo, oxr=oxr, $
  oyr=oyr, title='Unblended Lines + Instrument Features'

;NOTE:  uncorrected bug -- first two unblended lines seem to get
;       label locations forcibly set to 0.
     -------------------------------------------------------------------

4. IDENTIFY LINE FEATURES

   See http://hea-www.harvard.edu/PINTofALE/doc/LINEID.howto for a
   detailed description on how to use LINEID(). Here we only present a
   sample call and a sample set of IDs, for use in examples below. The
   following calls require that the variables defined in the
```

```
   initialization section are present.
lineid_idstr = lineid(data_wvl, data_spec, stor=rd_line_linstr, $
  n_e=!EDENS, dbdir=!LDBDIR, chifil=1, eqfile=!IONEQF, chidir=!CHIDIR, $
  DEM=!DEM*EXPTIM_V711TAU, abund=!ABUND, effar=effar, wvlar=wvlar, $
  verbose=!VERBOSE)

;NOTE 1:
;       since RD_LINE_LINSTR already exists with the correct emissivities,
;       there is no need to read them all in all over again, which LINEID
;       does automatically otherwise.  The RD_LINE()/FOLD_IONEQ() keywords
;       such as N_E, DBDIR, EQFILE, etc. are used only if STOR is not
;       properly defined.

;NOTE 2:
;       The sample IDs are in an IDL save file, at !ARDB/example_2.save:
;save,file=!ARDB+'/example_2.save',lineid_idstr

;       restore stored values to continue with current demo
;
restore,!ARDB+'/example_2.save',/verbose
     -------------------------------------------------------------------

5. DERIVE LINE FLUXES

   See http://hea-www.harvard.edu/PINTofALE/doc/FITLINES.howto for a demo
   of fitting line profiles. Here, as above, we only present a sample call
   proceeding from previously derived line identifications, for use in
   further examples below.
;       need do only the following to restore all necessary variables
;       (uncomment if cold start)
;
;.run initale
;restore,!ARDB+'/demo_v711Tau.save'
;restore,!ARDB+'/example_2.save'

;       extract some useful information out of the ID structure

;       first, the number of independent line features --
;
fitlines_pos=lineid_idstr.WVL & ncomp=n_elements(fitlines_pos)

;       then, the expected fluxes --
;
fitlines_flx=fltarr(ncomp)
for i=0L,ncomp-1L do fitlines_flx[i]=total(lineid_idstr.(i+1).FLUX)
```

```
;       set the widths to some reasonable number
;
fitlines_wdt=0.01

;       choose a function type for the line profile
;
fitlines_type='beta=1.8'

;       make labels for each feature, so that it will be easy to keep track
;
fitlines_epithet=strarr(ncomp)
all_labels = idlabel(lineid_idstr, j, /Wbeda)
for i=0L,n_elements(j)-1L do $
  fitlines_epithet[j[i]]=fitlines_epithet[j[i]]+all_labels[i]+' '

;       fit
;
fitlines_fitstr = fitlines(data_wvl, data_spec, ysig=data_spec_err, $
  pos=fitlines_pos, flx=fitlines_flx, wdt=fitlines_wdt, $
  perrp=fitlines_perrp, perrm=fitlines_perrm, perrc=fitlines_perrc, $
  ferrp=fitlines_ferrp, ferrm=fitlines_ferrm, ferrc=fitlines_ferrc, $
  werrp=fitlines_werrp, werrm=fitlines_werrm, werrc=fitlines_werrc, $
  thaw=fitlines_thaw, type=fitlines_type, epithet=fitlines_epithet, $
  ties=fitlines_ties, conlev=fitlines_conlev, consig=fitlines_consig, $
  /dumb, verbose=!verbose)

;NOTE:
;       the state of the fit is in the IDL save file (dumped from
;       within the GUI) !ARDB+'example_3.save'
;       the output is in an IDL save file
;save,file=!ARDB+'/example_4.save',fitlines_fitstr

;       restore stored values to continue with current demo
;
restore,!ARDB+'/example_4.save',/verbose
     ------------------------------------------------------------------

6. COMPUTE EMISSION MEASURES

;       need do only the following to restore all necessary variables
;       (uncomment if cold start)
;
;.run initale
;restore,!ARDB+'/demo_v711Tau.save'
```

```
;restore,!ARDB+'/example_2.save'
;restore,!ARDB+'/example_4.save'

;       here we will use the measured fluxes for the identified lines
;       in conjunction with the emissivity profiles to derive upper limits
;       to the emission-measure distribution at each temperature.

potem_tool_EM = potem_tool(lineid_idstr, fitlines_fitstr.flx, logT=!LOGT, $
  abund=!ABUND, multid=potem_tool_multid, effar=effar, wvlar=wvlar, $
  NH=!NH, He1=!He1, HeII=!HeII, verbose=!verbose)

;       plot the output EM curves
;
plotem, !LOGT, potem_tool_EM, idstr=lineid_idstr, multid=potem_tool_multid, $
  xr=[6,8], lsiz=1

;NOTE:
;       the emission measures are in the IDL save file !ARDB+'example_5.save'
;save,file=!ARDB+'example_5.save',potem_tool_EM,potem_tool_multid

;       restore stored values to continue with current demo
;
restore,!ARDB+'/example_5.save',/verbose
     ------------------------------------------------------------------

7. CREATE A LaTeX TABLE OF THE IDs

;       requires the outputs of feature identifications and
;       line profile fitting.  (uncomment if cold start)
;
;.run initale
;restore,!ARDB+'example_2.save'
;restore,!ARDB+'example_4.save'

;       first, update the locations of the features and the
;       line fluxes according to the measured (fitted) values
;
fitlines_flxerr=0.5*(fitlines_fitstr.ferrp+fitlines_fitstr.ferrm)
lineid_idstr = updatid(lineid_idstr, fitlines_fitstr.flx, $
  fitlines_fitstr.pos, flxerr=fitlines_flxerr)
help, cat_id(lineid_idstr)

;       now write out the updated information to a LaTeX file
;
tekku, lineid_idstr, texfil=!ARDB+'/example'
```

```
;       the resulting LaTeX file may be compiled and displayed
;
cd, '/tmp'
spawn, 'latex '+!ARDB+'/example'
spawn, 'xdvi example'
     -----------------------------------------------------------------
```

`pintofale()head.cfa.harvard.edu (VK)`

4.1.2 .par Session

Using .par files

The `.par` files are invoked on the `IDL` command line using the `@` symbol, eg:

```
  @rd_line.par
```

To do this, the user must have the `.par` file in the current working directory. All the reference .par files included in the `PINTofALE` release are kept in the `.../ardb/` directory, so they should be copied from here to the working directory when needed. The idea is that the user can either edit the contents of the `.par` to change the input variable names as required, or else set up any input variables according to the assumed names of the variables in the `.par` file prior to running it.

Looking inside `rd_line.par`, that calls `rd_line()` to read in line emissivity data from one of the databases:

```
; (VK, DecMM)
message,'Running RD_LINE()',/info
rd_line_linemis = rd_line( $
  atom ,$ ;no need to define this variable
  pres    = !GASPR ,$
  logP    = !LOGPR ,$
  n_e     = !EDENS ,$
  wrange  = [!WMIN,!WMAX] ,$
  dbdir   = !LDBDIR ,$
  wvl     = rd_line_wvl ,$
  logT    = rd_line_logT ,$ ;should be same as !LOGT
  z       = rd_line_z ,$
  ion     = rd_line_ion ,$
```

```
  jon     = rd_line_jon ,$
  src     = rd_line_src ,$
  desig   = rd_line_desig ,$
  econf   = rd_line_econf ,$
  fstr    = rd_line_linstr ,$
  allah   = 0 ,$
  help    = 0 ,$
  verbose = !VERBOSE )
```

Two things to notice are the use of the system variables for standard quantities such as `!EDENS` (electron density), `!WMIN`, and `!WMAX` (the wavelength interval specifiers), and that the output variable names have been designed to a convention (not always rigorously applied) where the variable name is made up from the routine name (`rd_line`) and the keyword name. This system was designed so as to be able to go from one `.par` to another with minimal editing of the `.par` or re-naming and re-assigning of variables. E.g., in order to fold the ionization balance into the line emissivity data read by `rd_line()`, `fold_ioneq.par` can be used, which contains

```
; (VK, DecMM)
message,'Running FOLD_IONEQ(); requires RD_LINE() output',/info
fold_ioneq_linemis = fold_ioneq( $
  rd_line_linemis ,$ ;output of RD_LINE()
  rd_line_Z ,$ ;output of RD_LINE()
  rd_line_JON ,$ ;output of RD_LINE()
  logT    = rd_line_logT ,$ ;output of RD_LINE()
  tmax    = fold_ioneq_tmax ,$
  trng    = fold_ioneq_trng ,$
  level   = 0.1 ,$
  chifil  = 1 ,$
  chidir  = !CHIDIR ,$
  eqfile  = !IONEQF ,$
  verbose = !VERBOSE )
```

where the line emissivity variable in `fold_ioneq.par` is *assumed* to be `rd_line_linemis`, which is the output of `rd_line.par`. Of course, it is always possible to edit the `.par` file if `fold_ioneq.par` is to operate on an alternative variable name, or if output is desired in variables with names different from those specified in the `.par`. Alternatively, the assumed variables in the `.par` files can be assigned to the variables one wished to use, without the need for editing the `.par`, for example:

```
@rd_line.par
test_linemis = rd_line_linemis
; save the variable so that rd_line
 . ; can be re-run with different settings
 .
rd_line_linemis = test_linemis  ; re-assign test_linemis to input variable
@fold_ioneq.par
```

This is the approach used in the `.par` example below.

The Example

```
       Example session using .par files and less explicit plotting etc

   This annotated example walks a user through a PINTofALE session using
   predominantly .par files where available. It is similar to the explicit
   session, but is designed to be simpler, less wordy, less explicit
   plotting, etc. During this run, the user will
    1. read in line and continuum emissivity databases,
    2. compute a theoretical spectrum,
    3. spectral browsing using SHOW_LINE
    4. identify line features in the observed spectrum,
    5. derive line fluxes by fitting line profiles,
    6. compute emission measures appropriate for the derived fluxes, and
    7. create a LaTeX table of the identifications
     ------------------------------------------------------------------

0. COPY .pars, INITIALIZE SETTINGS, AND LOAD DATA

# copy all the .pars from the ardb directory into working directory -
# in practice one can do this as each .par is needed.

cp /foo/bar/PoA/ardb/*.par ./.

#       start IDL
idl

;       set up the PoA environment

.run initale

;NOTE:
;       if INITALE fails, run the script
;       @PoA_constructor
```

```
;       load the data and instrument files
restore,!ARDB+'/demo_v711Tau.save'
;       put data into standard variables for observed spectrum
;       used in .par files, and then plot them; first counts spectrum

data_wvl=LAM_M1P_V711TAU & data_spec=SPC_M1P_V711TAU
wvlar=WVLAR_M1P & effar=EFFAR_M1P
!LOGT=LOGT_V711TAU & !DEM=DEM_V711TAU

plot,data_wvl,data_spec,psym=10,xrange=[5.,20.]
plot,wvlar,effar,xr=[5.,20.]            ; effective area
plot,!LOGT,!DEM,psym=10,/ylog   ; Rough DEM from EUVE Analysis
     ------------------------------------------------------------------

1. READ IN THE LINE AND CONTINUUM DATABASE

;       set the wavelength range of interest

!wmin = min(data_wvl)
!wmax = max(data_wvl)

;       read in the line emissivity database; this .par writes, by default,
;       emissivities into out_linemis an id structure into out_linstr
;       Of course, these names can be edited at will in the .par
;       [1e-23 ergs cm^3 s^-1]

@rd_line.par

;       fold in line emissivities with ion-balance info
;       (needed for CHIANTI and SPEX emissivities that are stored
;       independently from ion balance)

@fold_ioneq.par

; put the corrected emissivities into the line emissivity structure
; note that rd_line_linemis will still have emissivites without ion
; balance folded in.

rd_line_linstr.LINE_INT = fold_ioneq_linemis

;       Read in two sets of densities and manually interpolate to
;       desired density.  Making interpolation weights is same as for
;       explicit example; for the rest, just remember to change !edens
;       to the values required, and remember to change back to desired default
```

```
;       value.  NOTE: in practise this should not be necessary unless
;       synthesis of very density-sensitive features if required at a
;       very specific density.  Even then, this interpolation is only
;       linear in log10.

n_e=5e10 & n_e1=1e10 & n_e2=1e11
lne=alog10(n_e) & lne1=alog10(n_e1) & lne2=alog10(n_e2)
weight1=(lne-lne1)/(lne2-lne1) & weight2=(lne2-lne)/(lne2-lne1)

!edens=n_e1
@rd_line.par
rd_line_linemis1=rd_line_linemis
!edens=n_e2
@rd_line.par
rd_line_linemis2=rd_line_linemis
!edens=n_e

rd_line_linemis=10.^(weight1*alog10(rd_line_linemis1>1e-30)+$
  weight2*alog10(rd_line_linemis2>1e-30))

; fold ion balance again and put into structure - again, note that
; rd_line_linemis will still have emissivites without ion balance folded
; in

@fold_ioneq.par
rd_line_linstr.LINE_INT = fold_ioneq_linemis

;       read in the continuum emissivity database; ion balance already
;       included in these; will be on wavelength grid out_cwgrid
;       [1e-23 ergs cm^3 s^-1 Ang^-1]

@rd_cont.par
     -------------------------------------------------------------------

2. COMPUTE THEORETICAL SPECTRUM

;NOTE:
;       there are many different programs in PINTofALE (e.g., LINESPEC,
;       LINESPEC_EM, MAKE_SPECTRUM, etc.) that will allow one to generate
;       a theoretical spectrum.  The basic scheme carried out is to first
;       compute observed line fluxes from the line emissivities and bin
;       them into a spectrum, and add the continuum spectrum, derived by
;       computing the fluxes and rebinning to the same wavelength grid as
;       the line spectrum.  This scheme is shown below using some .par's,
;       though the wavelength grid first needs to be explicitly defined.
```

```
;       first define a wavelength grid to hold the spectrum.  Note
;       that one could simply use the grid of the observed spectrum,
;       though it is useful to then get a grid of the bin boundaries
;       since these are used by programs such as HASTOGRAM.
;       eg
;  data_dwvl=data_wvl[1:*]-data_wvl
;  data_wvlb=[data_wvl-data_dwvl/2. , data_wvl[n_elements(data_wvl)-1] $
;              + data_dwvl[n_elements(data_dwvl)-1]/2]

sp_NBIN=20000L  ;number of bins
sp_DW=(!WMAX-!WMIN)/sp_NBIN     ;bin width
sp_wgrid=findgen(sp_NBIN+1L)*sp_DW+!WMIN

;       compute the line fluxes
;       emissivities are [ergs cm^3/s], DEM is [/cm^5], effective
;       area is [cm^2].  the [ergs] are converted internally to [ph],
;       so the output of LINEFLX has the units [ct/s]
;NOTE:  could use lineflx+ismtau.par here to do flux and ism
;       absorption all at once.  Note, in this .par example we multiply the
;       combined line+continuum spectrum by the exposure time right at
;       the end, unlike in the explicit example.

@lineflx.par

;       bin the line fluxes into a spectrum.  hastogram is an improved
;       general histogram routine, much faster than the IDL histogram

linspec=hastogram(abs(rd_line_WVL),sp_wgrid,wts=lineflx_flx)

;       compute continuum fluxes (is actually a flux density), then multiply
;       by bin width. contflx.par actually calls
;       lineflx, but works consistently for continuum too.  Output
;       in contflx_flx

@contflx.par
rd_cont_CDW=rd_cont_wvl[1:*]-rd_cont_wvl
contflx_flx=contflx_flx*rd_cont_CDW

;       rebin to defined grid sp_grid - another completely general PoA utility

conspec=rebinw(contflx_flx,rd_cont_wvl,sp_wgrid,/perbin)

;       add the line and continuum spectra
```

```
spec=linspec+conspec

;       correct for ISM absorption.  Remember ISM parameters are set
;       up in variables !NH,!FH2,!HE1,!HEII that are used in
;       ismtau.par and output is written to out_tau
;       Need to set in_wvl variable to wavelength grid

in_wvl=sp_wgrid
@ismtau.par
spec=spec*exp(-ismtau_tau)

;       Now convert to total counts and plot the result - without axis titles!

spec=spec*EXPTIM_V711TAU

plot,sp_wgrid,spec,psym=10,xr=[5.,30.],/xs,yr=[1e-2,5e3],/ys,/ylog
setkolor,'red',1 & setkolor,'yellow',2
oplot,sp_wgrid,linspec,color=1,psym=10
oplot,sp_wgrid,conspec,color=2,psym=10

;       convolve with lorentzian of half-width=5*bin-width to simulate LRF
;       This is an example of a "general utility" PoA routine that is
;       not coronal plasma specific.  Use "beta model" with exponent 1.8

lrfspec=voorsmooth(spec,5.,type='beta=1.8')

;       plot the result and unconvolved spectrum

plot,sp_wgrid,spec,psym=10,xr=[5,8],/xs,/ylog
oplot,sp_wgrid,lrfspec,color=1,psym=10,thick=2

;NOTE:
;       these variables have been saved in !ARDB/example_1.save:
;save,file=!ARDB+'/example_1.save',sp_wgrid,spec,lrfspec,linspec,conspec

; The above will give a "complete" spectrum with all the lines in the
; database.  To compute, eg, just the spectrum of Ne with no continuum
; is easy by selecting only the lines from elements with atomic number
; = 10 (note that using the full wavelength range of sp_wgrid is a bit
; inefficient here though).

iNe=where(rd_line_linstr.z eq 10)
linespec_Ne=hastogram(abs(rd_line_WVL[iNe]),sp_wgrid,wts=lineflx_flx[iNe])
lrfspec_Ne=voorsmooth(linespec_Ne,5.,type='beta=1.8')
```

```
plot,sp_wgrid,linespec_Ne,psym=10,xr=[9,14],/xs
oplot,sp_wgrid,lrfspec_Ne,color=1,psym=10,thick=2
; zoom in on "triplet"
plot,sp_wgrid,linespec_Ne,psym=10,xr=[13.3,13.8],/xs
oplot,sp_wgrid,lrfspec_Ne,color=1,psym=10,thick=2


; Yet another way to generate a line spectrum from scratch is to use
; LINESPEC - a routine built up from the individual routines called
; above to arrive at a spectrum.  See linespec.par for the I/O
; used and assumed (eg, !DEM, !ABUND, !WMIN,!WMAX, effar, wvlar etc).
; Output of LINSPEC is a flux density, so need to x by binsize to get
; flux or count/bin

io_ww=sp_wgrid
@linespec.par

; This should create an output linespec_spec that, provided none of the
; system or other standard variables have been modified, should be the
; same as linspec created above after multiplying by bin width:

plot,linspec,linespec_spec*sp_dw,psym=3,/xlog,/ylog
     -------------------------------------------------------------------

3. SPECTRAL BROWSING

; find locations of lines of a given element and ionisation stage,
; say, all lines of Ne. - recall iNe defined above by
; iNe=where(rd_line_linstr.z eq 10)
; First set inputs to showline in_x,in_z,in_ion (latter two define the
; labels and are not required.  Can also specify in_y to set y level
; for labels.  If in_y is not set, labels appear at 2/3 y range.

in_x=abs(rd_line_linstr.wvl[iNe])
in_z=rd_line_linstr.z[iNe]
in_ion=rd_line_linstr.ion[iNe]
@show_line.par

; In show_line.par, a single middle mouse button click will place the
; line labels to their default positions (will also toggle log/linear
; axis).  Can test out features of
; pickrange by zooming in on a rectangular region drawn with the
; click-and-drag of the right mouse button (followed by left button
; click to enact the zoom); when an interesting feature has
; been found, get cursor control with a left button click, then dump a
```

```
; postscript hardcopy by typing h (can toggle colour, encapsulated
; or full landscape by typing c, e, and l), then = to dump the file

; To get a rough first idea of line ID's for prominent lines , just
; select lines that are stronger than 1/10 the flux of the strongest
; line:

istrong=where(lineflx_flx/max(lineflx_flx) gt 0.1)
in_x=abs(rd_line_linstr.wvl[istrong])
in_z=rd_line_linstr.z[istrong]
in_ion=rd_line_linstr.ion[istrong]
@show_line.par

; still in show_line (actually within pickrange), try, eg, browsing
; through the spectrum by zooming in on a narrow wavelength range to
; the short wavelength end (right button click & drag, then click with
; left button to select), then click again with left button to get
; keyboard control; now > and < scroll through the spectrum to right
; and left.

; One more: Fe XVII lines, but only the stronger ones again, and
; scaling the y position of the labels according to the theoretical
; fluxes from lineflx, using ~200 as peak of Fe XVII resonance line at
; 15.015 AA, and adding to very rough continuum derived from smoothing
; the spectrum

ife17=where(rd_line_linstr.z eq 26 and rd_line_linstr.ion eq 17)
ifexvii=where(rd_line_linstr.z eq 26 and rd_line_linstr.ion eq 17 and $
  lineflx_flx/max(lineflx_flx[ife17]) gt 0.05)
in_x=abs(rd_line_linstr.wvl[ifexvii])
in_z=rd_line_linstr.z[ifexvii]
in_ion=rd_line_linstr.ion[ifexvii]
in_y=200.*lineflx_flx[ifexvii]/max(lineflx_flx[ifexvii]) $
 + interpol(smooth(data_spec,1000),data_wvl,in_x)
@show_line.par

; Note that zooming in here does not change y label location because y
; positions are fixed by in_y

; find density sensitive lines using LSD.  This routine needs to read
; in the emissivities for different densities in order to find which
; lines have changed flux.  This example picks lines whose fluxes
; change by a factor of 5 or more, and that are 1/100 or stronger
; than the strongest density sensitive lines at any of the densities
; investigated.  Do this using SPEX emissivities (but many fewer density
```

```
; sensitive lines than CHIANTI)

in_ceiling=5.
in_flor=1.e-2
in_edens=[1.e9,1.e13]
!LDBDIR='$SPEX'
@lsd.par
delvar,in_y
in_x=lsd_wvl
in_z=lsd_z
in_ion=lsd_ion
@show_line.par
!LDBDIR='$CHIANTI'              ; re-set !LDBDIR to CHIANTI for later

; find lines that are relatively unblended - less than 20%.  First set the
; "resolution" - the +/- delta wavelength from line centre within
; which to consider neighbouring lines as blended.  Output in blend_frac

in_dwvl = 0.015         ; in AA, reasonable rough number for MEG
@bland.par
iblendfree=where(blend_frac gt 0.8 and lineflx_flx/max(lineflx_flx) gt 0.1)
in_x=abs(rd_line_linstr.wvl[iblendfree])
in_z=rd_line_linstr.z[iblendfree]
in_ion=rd_line_linstr.ion[iblendfree]
@show_line.par

; Now run smudge to find instrument features and use in showline too,
; first selecting the instrument config (this is accomplished by
; logical contructions using in_keys, in_only and in_nisi in
; smudge.par - yes, it is not so obvious but was done that way to
; avoid a huge and repetitive smudge.lst file

in_keys='HRMA MEG'
@smudge.par
@show_line.par
     ------------------------------------------------------------------

4. IDENTIFY LINE FEATURES

;       See LINEID.howto for a detailed description on how to use LINEID.
;       Here we only present a sample call and a sample set of IDs, for
;       use in examples below.
;       As mentioned in explicit demo, rd_line can of course work with
;       emissivities already read in.  This is controlled by the STOR
;       keyword.  In lineid.par, stor=out_linstr and if out_linstr is
```

```
;       a valid structure from rd_line then rd_line and fold_ioneq will not be
;       called.  Otherwise, these will be called and output will be put in
;       out_linstr.  In this case, calls to rd_line and fold_ioneq are
;       controlled by their keywords in lineid.par

@lineid.par

; Try playing with the features of lineid - eg zoom in on a region
; using the standard PICKRANGE cursor or keyboar commands; select a
; line using a click and drag over the line peak; play with changing
; the parameters in the pop-up GUI listing potential line ID's

;NOTE:
;       The sample IDs are in an IDL save file, at !ARDB/example_2.save:
;save,file=!ARDB+'/example_2.save',out_idstr
     ------------------------------------------------------------------

5. DERIVE LINE FLUXES

;       see FITLINES.howto for a demo of fitting line profiles.  Here, as
;       above, we only present a sample call proceeding from previously
;       derived line identifications, for use in examples below.

;       extract some useful information out of the ID structure
;
; To start from here without going through rest of example session,
; just do
;
;.run initale
;restore,!ARDB+'/demo_v711Tau.save'
;restore,!ARDB+'/example_2.save'

;       set the widths to some reasonable number and
;       choose a function type for the line profile

fitlines_wdt=0.01
fitlines_type='beta=1.8'

;       fit -  fitlines is very powerful in terms of ability to tie
;       together variables during fitting.  See FITLINES.howto in the
;       doc directory for more information.  One very useful feature
;       is the ability to set up ties between parameters on calling,
;       so that, eg all the line widths are tied together.
;       Here we are calling lineid_fitlines.par that sets up the line
;       parameters from the ID structure output from LINEID
```

```
@lineid_fitlines.par

;NOTE:
;       the state of the fit is in the IDL save file (dumped from
;       within the GUI) !ARDB+'example_3.save'
;       the output is in an IDL save file
;save,file=!ARDB+'/example_4.save',fstr
     ------------------------------------------------------------------

6. COMPUTE EMISSION MEASURES

;.run initale
;restore,!ARDB+'/demo_v711Tau.save'
;restore,!ARDB+'/example_2.save'
;restore,!ARDB+'/example_4.save'

;       here we will use the measured fluxes for the identified lines
;       in conjunction with line emissivities to derive upper limits
;       to the emission-measure distribution at each temperature.
;       ("Carole Jordan type" plot!)

in_flx=fitlines_fstr.flx
@potem_tool.par

;       plot the output EM curves

@plotem.par

;NOTE:
;       the emission measures are in the IDL save file !ARDB+'example_5.save'
;save,file=!ARDB+'example_5.save',out_LOGT,out_EM,out_multid
     ------------------------------------------------------------------

7. CREATE A LaTeX TABLE OF THE IDs

;       first, update the locations of the features and the
;       line fluxes according to the measured (fitted) values
;       output from updateid is updateid_idstr

@fitlines_updateid.par

;       now write out the updated information to a LaTeX file (this
;       part is identical to the explicit demo except for idstr name)
```

```
tekku,updateid_idstr,texfil=!ARDB+'/example'

;       the resulting LaTeX file may be compiled and displayed
cd,'/tmp'
spawn,'latex /example'
spawn,'xdvi example'
     ------------------------------------------------------------------
```

pintofale()head.cfa.harvard.edu (VK)

4.1.3 RD_LIST Rough Guide

From the file `doc/RD_LIST.howto`

```
READING IN CUSTOM LINE LISTS                              (JJD-JanMMI)
======================================================================

; Playing with line lists, using RD_LINE(), RD_LIST() and CAT_LN():
; Quick Example
;
; The function RD_LIST() can read in an ascii line list and from this
; can retrieve the appropriate emissivity data from any of the supported
; databases in the same way that RD_LINE() does.  RD_LIST() was designed so
; that the user can make up a "custom" line list that cannot easily be
; done with RD_LINE().
;
; If one simply wants to read in all lines from a
; given element, or set of elements or ions, and from a single database,
; this is trivial using RD_LINE() by specifying the desired species in the
; ATOM string, eg,
;   in_atom=['He', 'c 5', 'N V', 's_8']
; to illustrate the flexibility of specification.
;
; Now, if one is only interested in relatively few lines, and
; would like to use different database sources for different lines,
; RD_LIST() can be used.

; First generate a RD_LIST() understandable line list, either using
; CAT_LN() (or "by hand"; see the RD_LIST.PRO documentation for the
; straightforward format) and the lines already read in by RD_LINE().
; Let's look at 13-14 AA:

i13_14=where(rd_line_linstr.wvl gt 13. and rd_line_linstr.wvl le 14)
cat_ln_linstr=cat_ln(rd_line_linstr,pick=i13_14,flst='chianti13_14',/comm)
```

```
; now switch databases and read the SPEX DB for 13-14 AA; remember to
; save the CHIANTI structure already read in by RD_LINE, because the
; standard rd_line_linstr will get over-written unless changed in the
; .par file.

rd_line_linstrch=rd_line_linstr
!LDBDIR = '$SPEX'
!WMIN=13. & !WMAX=14.
@rd_line.par

; print them all out

cat_ln_linstr=cat_ln(rd_line_linstr,flst='spex13_14',/comm)

; Change ascii line list how one likes, eg, swap SPEX Ne VIII lines for
; CHIANTI's more extensive list and re-read SPEX, with CHIANTI Ne VIII
; unix_prompt$ grep -v 'Ne VIII' spex13_14 > temp1
; unix_prompt$ grep 'Ne VIII' chianti13_14 > temp2
; unix_prompt$ cat temp1 temp2 > mix13_14

rd_list_linstr=rd_list('./mix13_14')

; Note that this example is just to illustrate the ease with which the
; input line list can be "mixed and matched".  The resulting
; RD_LIST_LINSTR in this particular example could also have been
; generated by selecting out Ne VIII structure elements from the CHIANTI
; structure using the IDL WHERE command into a new structure, doing the
; inverse operation on the SPEX structure, then using CAT_LN:
;
;     linstrAB=cat_ln(linstrA,linstrB)
```

One thing to add is that `rd_list` is very much slower than `rd_line`, and so is better suited to shorter line lists tailored to a particular analysis (e.g., looking the same set of lines in different spectra) than to extensive spectral simulations. For the latter, when mix-and-match is required, it can often be achieved by selecting from different line structures and then cat'ing line structures together using `cat_ln()`.

4.1.4 `LINEID` Rough Guide

From the file `doc/LINEID.howto`

```
How to use LINEID
```

```
=================

LINEID is an interactive function to identify line features in a spectrum.
It returns an IDL structure containing the identified features, on which
other functions such as CAT_ID, UPDATID, ID2EMIS2ID, etc. operate.

Below are various examples of how to use LINEID.


0. CALLING SEQUENCE
-------------------


   The most general calling sequence is

     idstr=lineid(lamda,spec,elem,wrng=wrng,wshift=wshift,locmax=locmax,$
        best=best,topX=topX,batch=batch,stor=stor,oldid=oldid,omatch=omatch,$
        n_e=n_e,logP=logP,pres=pres,dbdir=dbdir,/allah,chifil=chifil,$
        eqfile=eqfile,chidir=chidir,dem=dem,abund=abund,effar=effar,$
        wvlar=wvlar,sigmay=sigmay,xsize=xsize,ysize=ysize,wid=wid,$
        dynrng=dynrng,oxr=oxr,oyr=oyr,markp=markp,marks=marks,marko=marko,$
        /nuthin,/noday,/notime,/nouser,/nopack,stacol=stacol,stasiz=stasiz,$
        stathk=stathk)

   See the documentation for a description of the variables.


1. BATCH MODE/COMMAND LINE
--------------------------


   To simply list potential matches of a single feature on the command line,
   a la CIAO.Sherpa.GUIDE's identify() command, type:

   IDL> help,cat_id(lineid(WAVELENGTH))

   for example,

   IDL> help,cat_id(lineid(22.1))

   will read through the default line database (CHIANTI) and return with
   a default set number of possible matches (20) within a default wavelength
   range (+-0.221)

   % RD_LINE:      No line information available for H
```

```
..
% RD_LINE:      No lines in this wavelength range
.% RD_LINE:     No line information available for Li
% RD_LINE:      No line information available for Be
% RD_LINE:      No line information available for B
..
% RD_LINE:      No lines in this wavelength range
.............% RD_LINE:         No line information available for F
..
% RD_LINE:      No lines in this wavelength range
...
% RD_LINE:      No lines in this wavelength range
...
% RD_LINE:      No lines in this wavelength range
...
% RD_LINE:      No lines in this wavelength range
.........
% RD_LINE:      No lines in this wavelength range
...
% RD_LINE:      No lines in this wavelength range
...
% RD_LINE:      No lines in this wavelength range
...................% RD_LINE:   No line information available for Sc
..
% RD_LINE:      No lines in this wavelength range
.% RD_LINE:     No line information available for V
..
% RD_LINE:      No lines in this wavelength range
...
% RD_LINE:      No lines in this wavelength range
.........
% RD_LINE:      No lines in this wavelength range
...
% RD_LINE:      No lines in this wavelength range
.% RD_LINE:     No line information available for Cu
..
% RD_LINE:      No lines in this wavelength range
.

 WVL [Ang]       MATCH      Z ION      FLUX          +-     Tmax
0-----------------------------------------------------1
   22.1000:    22.1012      O VII     0.17841 +-   0.000    6.30
   22.1000:    22.1020       O VI  2.0272e-06 +-   0.000    6.25
   22.1000:   -22.0970    Ca XVII  2.5012e-06 +-   0.000    6.70
   22.1000:   -22.1060     Ca XVI  5.1657e-07 +-   0.000    6.60
```

```
      22.1000:   -22.0939    Ar XVII  1.1069e-07 +-   0.000    7.30
      22.1000:    22.0931    Fe XXIV  3.9956e-08 +-   0.000    7.30
      22.1000:    22.1070    Ca XVII  3.1153e-08 +-   0.000    6.70
      22.1000:   -22.0870     Ca XVI  1.9131e-07 +-   0.000    6.60
      22.1000:    22.1140    Ca XVII  0.00064187 +-   0.000    6.70
      22.1000:    22.1174       O VI  2.9258e-14 +-   0.000    6.25
      22.1000:    22.1174       O VI  0.00064908 +-   0.000    6.20
      22.1000:    22.1200       O VI  4.7364e-15 +-   0.000    6.25
      22.1000:    22.1200       O VI  5.0908e-14 +-   0.000    6.25
      22.1000:    22.1200       O VI   0.0011296 +-   0.000    6.20
      22.1000:    22.1200       O VI  0.00010507 +-   0.000    6.20
      22.1000:    22.1210     Ca XVI  3.6417e-05 +-   0.000    6.60
      22.1000:    22.0780       O VI  3.7569e-05 +-   0.000    6.25
      22.1000:   -22.1230     Ca XVI  5.7742e-06 +-   0.000    6.60
      22.1000:    22.1230       O VI  7.2479e-06 +-   0.000    6.25
      22.1000:    22.0770       O VI  1.9083e-05 +-   0.000    6.25
   <Expression>    STRUCT    = -> <Anonymous> Array[1]

   The FLUX in this case would be in units of ph/s/cm^2, calculated
   for a flat DEM(T) or 10^14 cm^-5 over temperatures 10^4 - 10^8 K.

------------
1A. If not interested in all that informational messages, set VERBOSE=0
    for LINEID; further, to confine attention only to O lines, type:


    IDL> help,cat_id(lineid(22.1,'O',verbose=0))
    ......
     WVL [Ang]       MATCH      Z ION      FLUX         +-     Tmax
    0----------------------------------------------------1
       22.1000:    22.1012      O VII     0.17841 +-   0.000    6.30
       22.1000:    22.1020       O VI  2.0272e-06 +-   0.000    6.25
       22.1000:    22.1174       O VI  2.9258e-14 +-   0.000    6.25
       22.1000:    22.1174       O VI  0.00064908 +-   0.000    6.20
       22.1000:    22.1200       O VI   0.0011296 +-   0.000    6.20
       22.1000:    22.1200       O VI  0.00010507 +-   0.000    6.20
       22.1000:    22.1200       O VI  4.7364e-15 +-   0.000    6.25
       22.1000:    22.1200       O VI  5.0908e-14 +-   0.000    6.25
       22.1000:    22.0780       O VI  3.7569e-05 +-   0.000    6.25
       22.1000:    22.1230       O VI  7.2479e-06 +-   0.000    6.25
       22.1000:    22.0770       O VI  1.9083e-05 +-   0.000    6.25
       22.1000:    22.1240       O VI  1.5000e-05 +-   0.000    6.25
       22.1000:    22.0440       O VI  7.0315e-12 +-   0.000    6.30
       22.1000:    22.0440       O VI  6.2800e-05 +-   0.000    6.20
       22.1000:    22.0440       O VI  6.5579e-10 +-   0.000    6.30
```

```
       22.1000:    22.0440       O VI  6.1646e-08 +-   0.000    6.20
       22.1000:    22.0415       O VI  1.2613e-07 +-   0.000    6.20
       22.1000:    22.0415       O VI  1.1724e-05 +-   0.000    6.20
       22.1000:    22.0415       O VI  1.3417e-09 +-   0.000    6.30
       22.1000:    22.0415       O VI  1.3128e-12 +-   0.000    6.30
    <Expression>    STRUCT    = -> <Anonymous> Array[1]


------------
1B. To refine the identification further, type:


    IDL> help,cat_id(lineid(22.1,'O VII',verbose=0))
    ......

     WVL [Ang]       MATCH      Z ION      FLUX          +-     Tmax
    0-----------------------------------------------------1
       22.1000:    22.1012      O VII     0.17841 +-   0.000    6.30
    <Expression>    STRUCT    = -> <Anonymous> Array[1]

    It is also possible to use "O7", "o_7", etc. to denote O VII.


------------
1C. Rather than CHIANTI, to use the SPEX database computed for a constant
    electron density of 1e10 cm^-3 instead, type:


    IDL> help,cat_id(lineid(22.1,'O7',dbdir='$SPEX',n_e=1e10,verbose=5))
    % RD_LINE: Reading from database /foo/bar/PINTofALE/emissivity/spexD

    % RD_LINE:      Working on O7
    % RD_LINE: Reading from files:
    /foo/bar/PINTofALE/emissivity/spexD/O_wvl
    /foo/bar/PINTofALE/emissivity/spexD/O_tem
    /foo/bar/PINTofALE/emissivity/spexD/O_ion
    /foo/bar/PINTofALE/emissivity/spexD/O_jon
    /foo/bar/PINTofALE/emissivity/spexD/O_src
    /foo/bar/PINTofALE/emissivity/spexD/O_10.0
    /foo/bar/PINTofALE/emissivity/spexD/O_lvl
    /foo/bar/PINTofALE/emissivity/spexD/O_ecn
    ......
    ...
    % FOLD_IONEQ: Using ion balances from: ioneq/arnaud_rothenflug.ioneq
     WVL [Ang]       MATCH      Z ION      FLUX          +-     Tmax
```

```
0------------------------------------------------------1
   22.1000:    22.1000      O VII  0.00013270 +-   0.000    6.40
   22.1000:    22.1000      O VII     0.13018 +-   0.000    6.30
   22.1000:    22.1000      O VII    0.043981 +-   0.000    6.40
<Expression>    STRUCT    = -> <Anonymous> Array[1]
```

where the 3 components are due to inner-shell ionization, collissional excitation, and radiative-recombination.

1D. In order to avoid reading in the database each time, type:

```
IDL> idstr=lineid(22.1,'o_7',dbdir='$SPEX',chidir=!CHIDIR,eqfile=!IONEQF,$
n_e=1e10,verbose=0,stor=linstr)
```

where the keywords CHIDIR and EQFILE point to the appropriate ion balance file to use. The line database is read into the structure LINSTR. This can be reused without having to specify the keywords DBDIR, N_E, etc.:

```
IDL> help,cat_id(lineid(22.1,stor=linstr))
Emissivities cover the range:      22.1000 -       22.1000
current wavelength range:      21.8790 -       22.3210
% LINEID: WARNING: input emissivities do not cover the
% LINEID:          wavelength range of interest

 WVL [Ang]       MATCH      Z ION      FLUX          +-     Tmax
0------------------------------------------------------1
   22.1000:    22.1000      O VII  0.00013270 +-   0.000    6.40
   22.1000:    22.1000      O VII     0.13018 +-   0.000    6.30
   22.1000:    22.1000      O VII    0.043981 +-   0.000    6.40
<Expression>    STRUCT    = -> <Anonymous> Array[1]
```

(in this case, the warning is irrelevant; ignore it)
the details of each line can be summarized with:

```
IDL> help,cat_ln(linstr,/comm)
     Z ION  WVL [Ang]      FLUX     Tmax
0----------------------------------------------1
     O VII   22.1000  0.00013270  6.400   HE6      (II) ->
1----------------------------------------------2
     O VII   22.1000     0.13018  6.300   HE6      (EX) ->
2----------------------------------------------3
```

```
        O VII   22.1000    0.043981  6.400   HE6      (Recomb) ->
    <Expression>    STRUCT    = -> <Anonymous> Array[1]


------------
1E. Multiple features may also be identified in batch mode.  The keyword
    BATCH must be used in this case.  If the keyword BEST is also used,
    the program makes an informed decision based on the strength of the
    match as well as the nearness to the feature.


    IDL> help,cat_id(lineid([19.0,22.1],'O',/batch,/best))
    ......

     WVL [Ang]       MATCH      Z ION      FLUX          +-     Tmax
    0---------------------------------------------------1
       19.0000:    18.9671     O VIII     0.27836 +-   0.000    6.50
    1---------------------------------------------------2
       22.1000:    22.1012      O VII     0.17841 +-   0.000    6.30
    <Expression>    STRUCT    = -> <Anonymous> Array[1]


2. INTERACTIVE MODE
-------------------


   If a spectrum is available as an array as a function of a wavelength
   array, then multiple line features may be identified interactively.

   Here is an example walk-through, using an observed Chandra HETG/MEG
   (+1 order) spectrum of the RSCVn binary HR1099.

   IDL> .run initale
   IDL> restore,!ARDB+'/demo_v711Tau.save'
   IDL> help,name='*_[MV][17][P1]*'
   DEM_V711TAU     DOUBLE    = Array[81]
   EFFAR_M1P       FLOAT     = Array[8200]
   EXPTIM_V711TAU  FLOAT     =       136620.
   LAM_M1P_V711TAU DOUBLE    = Array[8192]
   LOGT_V711TAU    FLOAT     = Array[81]
   SPC_M1P_V711TAU INT       = Array[8192]
   SPE_M1P_V711TAU FLOAT     = Array[8192]
   WVLAR_M1P       FLOAT     = Array[8200]
   WW_M1P_V711TAU  DOUBLE    = Array[8193]
```

```
Here, SPC_M1P_V711TAU(LAM_M1P_V711TAU) are the observed counts
spectrum as a function of wavelength (WW_M1P_V711TAU are the
wavelength bin-boundaries);
SPE_M1P_V711TAU(LAM_M1P_V711TAU) are the Gehrel's approximation
to Poisson error on the counts;
DEM_V711TAU(LOGT_V711TAU) is an EUVE-based approximate Differential
Emission Measure distribution as a function of log(Temperature);
EFFAR_M1P(WVLAR_M1P) is the effective area as a function of
wavelength; and
EXPTIM_V711TAU is the exposure time for the observation.

Plot each of the above, for example, thusly:-
* The counts spectrum and errors
  plot,LAM_M1P_V711TAU,SPC_M1P_V711TAU,psym=10,xtitle='Wavelength ['+$
  string(byte(197))+']',ytitle='Counts/bin',title='HR 1099: HETG/MEG +1',$
  xrange=[5.,20.],yr=[0.1,1200],/ys
  for i=0L,n_elements(LAM_M1P_V711TAU)-1L do oplot,$
  LAM_M1P_V711TAU[i]*[1,1],SPC_M1P_V711TAU[i]+SPE_M1P_V711TAU[i]*[-1,1],$
  color=1 & setcolor,'red',1
* The effective area
  plot,WVLAR_M1P,EFFAR_M1P,xtitle='Wavelength ['+string(byte(197))+']',$
  ytitle='Effective Area [cm!u2!n]',title='HETG/MEG +1',xr=[5.,20.]
* The normalized spectrum
  tmp_effar=interpol(EFFAR_M1P,WVLAR_M1P,LAM_M1P_V711TAU)
  plot,LAM_M1P_V711TAU,SPC_M1P_V711TAU/tmp_effar/EXPTIM_V711TAU,$
  psym=10,xtitle='Wavelength ['+string(byte(197))+']',$
  ytitle='Counts cm!u-2!n s!u-1!n]',title='HR 1099: HETG/MEG +1',$
  xr=[5.,20.]
* The DEM
  plot,LOGT_V711TAU,DEM_V711TAU,psym=10,/ylog,xtitle='log!d10!n (T [K])',$
  ytitle='DEM [cm!u-5!n]',title='EUVE based approximate DEM for HR1099'

Note that if the DEM is not defined on the same temperature grid as the
emissivities, then it should be interpolated.  For example:
  oT=lindgen(21)*4 & tmp_LOGT=LOGT_V711TAU[oT] & tmp_DEM=DEM_V711TAU[oT]
  oplot,tmp_LOGT,tmp_DEM,psym=10,color=2 & setcolor,'yellow',2
  correct_logT=findgen(81)*0.05+4.
  new_DEM = mk_dem('interpolate',logT=correct_logT,$
    indem=tmp_DEM,pardem=tmp_LOGT)
  oplot,correct_logT,new_DEM,psym=10,color=3 & setcolor,'green',3

In this case, an appropriate calling sequence would be:

IDL> idstr=lineid(LAM_M1P_V711TAU,SPC_M1P_V711TAU,stor=stor_2,$
n_e=!EDENS,dbdir=!LDBDIR,chifil=1,eqfile=!IONEQF,chidir=!CHIDIR,$
```

```
DEM=DEM_V711TAU*EXPTIM_V711TAU,abund=!ABUND,$
effar=EFFAR_M1P,wvlar=WVLAR_M1P,verbose=!VERBOSE)
```

after writing a lot of informational messages (saying what element it is reading, etc.), the spectrum is plotted in a window and the user is invited to select a feature for identification. The selection is handled by the procedure PICKRANGE, which is described in detail elsewhere. For here, briefly note the following characteristics:

1. Select an area on the spectrum to zoom into using the right mouse button -- hold down and drag, say from (18,250) to (21,0). The numbers do not have to be precise.
2. Zoom into the selected area by clicking on the left button. (NOTE: clicking the middle mouse button will "unzoom" the plot; clicking on the right mouse button will pass control to the keyboard, which allows "scrolling" the plot window)
3. Select the line feature for identification using the left mouse button -- hold down and drag from approximately (18.5,150) to (19.5,150). This largest local maximum in the selected range will be the candidate for identification.
4. exit the interactive feature selection with a click (NO drag!) of the right mouse button.

At this point, a widget listing a number of matches will pop up. Select the best match(es) by clicking on the box in the left-most column. The candidate matches may also be sorted by other means, for example, by intensity (SORT->byIntensity), and the likely ID for this feature, O VIII 18.967, will rise to the top of the list. Click on the box at the left to select it as the ID.

Click on NEXT to continue with interactive feature identification, or click on QUIT to exit LINEID.

```
IDL> help,cat_id(idstr)
 WVL [Ang]       MATCH      Z ION      FLUX          +-     Tmax
0-------------------------------------------------------1
   18.9825:    18.9671     O VIII      357.54 +-   0.000    6.50
<Expression>    STRUCT    = -> <Anonymous> Array[1]
```

2A. It is possible to edit existing ID structures interactively. An existing set of IDs may be passed into LINEID using the keyword OLDID, and the manner of editing is controlled by the keyword OMATCH. For example, if you have spectra from different instruments

```
    of the same source, and have a set of IDs derived using one spectrum,
    you can use a value of OMATCH = -ve relative wavelength error to
    avoid IDing the same features multiple times.
```

4.1.5 `FITLINES` Rough Guide

From the file `doc/FITLINES.howto`

```
How to use FITLINES
===================

FITLINES is an interactive widget-based function to fit 1-dimensional
profiles to features in 1-D spectral data, Y[X].  In principle, it
only works with 3-parameter curve families (Gaussians, Lorentzians),
but in practice curves with extra parameters may be fit (Modified
Lorentzians, Rotationally Broadened Line) provided that the extra
parameters are held fixed.

Currently, the following functions are encoded via MK_3MODEL():

Function               TYPE definitions for MK_3MODEL()

Gaussian               G; GAUSSIAN; etc.
Lorentzian             L; LORENTZIAN; etc.
Beta-profile           B; BETA; BETA=<index>; BETA:<index>; BETA <index>; etc.
Asymmetric Slanted     SL; SLANT; SLANT=<angle>; SLANT=<angle>,<beta>; etc.
  Beta-profile
Rotation Broadened     R; ROT; ROT=<velocity>; etc.
  Gaussian
Broken Power-Law       PW; PW@<break>; PW=<break>; etc.
Identity               I; Ident; etc.
Sinusoid               SI; Sin=<phase>; etc.

FITLINES must be called as a function, and returns a structure containing
the best-fit positions, fluxes, widths, associated errors, etc.

To see the complete calling sequence, type
help,fitlines()
at the "IDL>" prompt.  The output lists all the parameters and keywords
accepted by FITLINES.  Do not panic: in the majority of the cases, only
a few of them have to be specified.  For a description on what each of
the variables mean, see
http://hea-www.harvard.edu/PINTofALE/PoA.html#FITLINES

An example walk-through for using FITLINES is given below.
```

This is by no means to be considered comprehensive, but
does exercise some of the major features of the GUI.

-- Start IDL

-- initialize the PINTofALE environment

```
    .run initale
```

-- restore a spectrum for fitting line profiles to (we will
fit beta-profiles to the Si XIII triplet at 6.7 AA):

```
    restore,!ARDB+'/demo_v711Tau.save',/verbose

    plot,LAM_M1P_V711TAU,SPC_M1P_V711TAU,psym=10,xr=[6.5,7.0],/xs,$
      xtitle='Wavelength ['+string(byte(197))+']',ytitle='Counts/bin',$
      title='HR 1099: HETG/MEG +1',yr=[0,400],/ys
```

-- initialize the input to FITLINES

```
    ;Specify that 3 model components are needed by declaring a 3-element
    ;array for the locations.  the locations need only be approximate.
    pos=fltarr(3)+6.5
```

NOTE: it is possible to start off with fewer components; new ones
may be added to the mix using the GUI.

```
    type='beta 1.8'   ;to fit beta-profiles of index 1.8
    wdt=0.05          ;set the initial values of the widths to 0.05
```

-- start up the GUI

```
    fstr=fitlines(LAM_M1P_V711TAU,SPC_M1P_V711TAU,ysig=SPE_M1P_V711TAU,$
    pos=pos,perrp=perrp,perrm=perrm,perrc=perrc,$
    flx=flx,ferrp=ferrp,ferrm=ferrm,ferrc=ferrc,$
    wdt=wdt,werrp=werrp,werrm=werrm,werrc=werrc,$
    thaw=thaw,type=type,epithet=epithet,ties=ties,$
    conlev=conlev,consig=consig,/dumb,verbose=!verbose)
```

NOTE: the /DUMB keyword ensures that FIT_LEVMAR does not stop every
so often to ask for directions. For those of you raised on XSPEC,
that is like setting query to yes while fitting.

-- display the full spectrum

```
   [PULL DOWN] VIEW -> SHOW

-- restrict X-axis range

   [PULL DOWN] XRANGE -> Keyboard
   [TYPE @ PROMPT] 6.5 7 <CR>
   [PULL DOWN] VIEW -> SHOW

   where "<CR>" represents a carriage-return.

   To also set the Y-axis range to something more useful,

   [PULL DOWN] YRANGE -> Current
   [PULL DOWN] VIEW -> SHOW

-- the initial guesses for the line locations are clearly very bad.
   reset them to more reasonable values at the parameter editing
   interface at center-left of the GUI.

   [TYPE IN BOX "Component"] 0
   [TYPE IN BOX "Position"] 6.65 <CR>
   [TYPE IN BOX "Label"] Si13 r <CR>
   [TYPE IN BOX "Component"] 1
   [TYPE IN BOX "Position"] 6.70 <CR>
   [TYPE IN BOX "Label"] Si13 i <CR>
   [TYPE IN BOX "Component"] 2
   [TYPE IN BOX "Position"] 6.75 <CR>
   [TYPE IN BOX "Label"] Si13 f <CR>

-- it is better to let the program figure out the best guesses
   for the initial values of the fluxes, so click on the buttons
   corresponding to the fluxes (parameters A2, A5, and A8) at the
   left of the center-right descriptor window, and then renormalize
   the fluxes

   [LEFT CLICK ON BUTTON FOR A2]
   [LEFT CLICK ON BUTTON FOR A5]
   [LEFT CLICK ON BUTTON FOR A8]
   [LEFT CLICK] RENORM

   the model profiles at this point look thus:

   [PULL DOWN] VIEW -> SHOW

-- there is also a fairly large continuum at this wavelength range.
```

```
   the continuum is best set "by hand".

   [PULL DOWN] CONTINUUM -> Piecewise
   [LEFT CLICK ON PLOT, AT APPROX (6.9,95)]
   (multiple clicks will allow a more complex definition)
   [LEFT CLICK IN PLOT WINDOW, BELOW X-AXIS]
   (get rid of default Poisson error on continuum estimation)
   [RIGHT CLICK WITHIN PLOT]
   (exit)
   [PULL DOWN] CONTINUUM -> ACCEPT
   (to adopt the newly defined continuum)   (VERY IMPORTANT!!!)

   [PULL DOWN] VIEW -> SHOW
   [LEFT CLICK] RENORM
   [PULL DOWN] VIEW -> SHOW

-- set up constraints on the parameters (these are essentially
   any legal IDL statement that modify the parameter values)

   these ensure that the widths of all the fitted lines are identical:
   [TYPE IN BOX "ADD"] A4 = A1 <CR>
   [TYPE IN BOX "ADD"] A7 = A1 <CR>

   these ensure that all fluxes are +ve:
   [TYPE IN BOX "ADD"] A2 = A2 > 0 <CR>
   [TYPE IN BOX "ADD"] A5 = A5 > 0 <CR>
   [TYPE IN BOX "ADD"] A8 = A8 > 0 <CR>

   far more complicated constraints may also be set.  for example,
   the following ensures that the flux of the intercombination line
   lies between a fifth to half that of the forbidden line
   [TYPE IN BOX "ADD"] A5 = (A5 < (A8/2.) > (0.2*A8))

   as another example, the following forces the location of the
   intercombination line to lie between that of the resonance and
   the forbidden lines
   [TYPE IN BOX "ADD"] A3 = (A3 > A0) < A6

-- Now thaw all the relevant parameters

   [LEFT CLICK ON BUTTON FOR A0]
   [LEFT CLICK ON BUTTON FOR A1]
   [LEFT CLICK ON BUTTON FOR A3]
   [LEFT CLICK ON BUTTON FOR A6]
```

```
   Note that A4 and A7 must be left frozen, because they are
   to have values identical to A1.  However, A3, A5, A6, and
   A7 are thawed, because even if strongly constrained, they
   must still be fit.

-- Fit

   [LEFT CLICK] FIT
   [PULL DOWN] VIEW -> SHOW

   which results in a reasonable fit (reduced chisq ~ 1.1)

   the widths may be -ve, which actually does not matter except for
   aesthetic reasons.  if -ve, reset them to +ve values using the
   parameter editing interface.

-- reduce the range over which the fit is carried out.  let the
   program guess at an appropriate range, which it does by looking
   at the values of all the thawed positions and widths.

   [PULL DOWN] XRANGE -> Guess

-- and fit again

   [PULL DOWN] VIEW -> SHOW
   [LEFT CLICK] FIT
   [PULL DOWN] VIEW -> SHOW

-- save the fit parameters internally

   [LEFT CLICK] SAVE

-- dump the current state to disk, for leisurely access via FITLINES_UNDUMP

   [LEFT CLICK] DUMP
   [TYPE @ PROMPT] /tmp/fitlines.save

-- Obtain 1-sigma chisq-surface-projected errors

   [LEFT CLICK] ERRORS

   NOTE: the error finding subroutine is very robust, but it may still
   crash (or go into a catatonic loop) when confronted with pathological
   error surfaces.  it is always possible to get back to the GUI in such
   cases by typing RETURN at the ERORS subroutine level.
```

```
-- fit to a new line, say the one at 6.2 AA

   [PULL DOWN] XRANGE -> RESET
   [PULL DOWN] XRANGE -> Keyboard
   [TYPE @ PROMPT] 5.5 7.0 <CR>
   [PULL DOWN] YRANGE -> Current

   Note the sharp drop at 6.5 AA, which is due to the fact that the
   continuum was defined only in the range 6.5-7.0 AA.  Choose a small
   interval in which to fit the new line:

   [PULL DOWN] XRANGE -> Cursor
   this will put you in PICKRANGE(), which is described elsewhere.
   All that matters here is to hold down the left button and drag
   it over the x-interval of interest.
   [LEFT DRAG, say from (6.1,100) to (6.3,100)]
   [RIGHT CLICK to exit PICKRANGE()]

   define a new model component
   [PULL DOWN] MODEL -> ADD Component
   hold down the left button and drag from the peak of the line
   to a point that denotes the width
   [DRAG, say from (6.19,350) to (6.2,150)]
   [CLICK to exit model definition]

   in order to fit only the new component to the selected line feature,
   all the other model components must be frozen
   [PULL DOWN] MODEL -> Freeze Unimportant
   [LEFT CLICK] RENORM

   the continuum must be redefined
   [PULL DOWN] CONTINUUM -> Piecewise
   to avoid using Poisson errors on the defined continuum,
   [LEFT CLICK BELOW X-AXIS]
   to adjust the continuum level, hold down the left button and drag
   to new level
   [LEFT DRAG, from say (6.2,95) to (6.2,90)]
   [RIGHT CLICK to exit]
   [PULL DOWN] CONTINUUM -> ACCEPT
   [LEFT CLICK] RENORM
   [PULL DOWN] VIEW -> SHOW

   [LEFT CLICK] FIT
   [PULL DOWN] VIEW -> SHOW
```

```
   which is again a reasonable fit, with a reduced chisq ~ 1.4

   [LEFT CLICK] ERRORS

-- to quit

   [LEFT CLICK] QUIT

-- all the useful quantities are in the output,
    IDL> help,fstr,/str
    ** Structure <39a2d0>, 21 tags, length=196952, refs=1:
       POS             FLOAT     Array[4]
       PERRP           FLOAT     Array[4]
       PERRM           FLOAT     Array[4]
       PERRC           FLOAT     Array[4]
       FLX             FLOAT     Array[4]
       FERRP           FLOAT     Array[4]
       FERRM           FLOAT     Array[4]
       FERRC           FLOAT     Array[4]
       WDT             FLOAT     Array[4]
       WERRP           FLOAT     Array[4]
       WERRM           FLOAT     Array[4]
       WERRC           FLOAT     Array[4]
       THAW            INT       Array[12]
       TYPE            STRING    Array[4]
       TIES            STRING    Array[7]
       EPITHET         STRING    Array[4]
       CONLEVX         DOUBLE    Array[8192]
       CONLEV          FLOAT     Array[8192]
       CONSIGX         DOUBLE    Array[8192]
       CONSIG          FLOAT     Array[8192]
       COMMENT         STRING    '<chisq=42.778772> <DUMB=1> <VERBOSE=10> {a4 ='...
    IDL> for i=0,3 do print,(fstr.pos)[i]+[-(fstr.perrm)[i],0,(fstr.perrp)[i]]
          6.65619      6.65661      6.65703
          6.69431      6.69571      6.69710
          6.74903      6.74935      6.74967
          6.18996      6.19036      6.19075
    IDL> for i=0,3 do print,(fstr.wdt)[i]+[-(fstr.werrm)[i],0,(fstr.werrp)[i]]
        0.0111397    0.0116787    0.0122177
        0.0111397    0.0116787    0.0122177
        0.0111397    0.0116787    0.0122177
        0.0115940    0.0122118    0.0128578
    IDL> for i=0,3 do print,(fstr.flx)[i]+[-(fstr.ferrm)[i],0,(fstr.ferrp)[i]]
          1063.90      1109.42      1154.95
```

```
152.692      184.313      215.934
612.465      651.284      690.103
1155.23      1205.77      1256.64
```

4.2 Making a Synthetic Spectrum using `MAKE_SPECTRUM`

It is not possible to represent the whole of `PINTofALE` by a single flow chart, because different analyses require different routines, or use of routines in slightly different ways. It can, however, be very useful to be able to see a flow chart for a particular analysis task that lays out the steps required and also to see the `PINTofALE` modules that need to be included. However, this is a very time consuming process, and we therefore provide only one such here, for the purpose of computing a synthetic spectrum. This algorithm has also been encoded in the script `make_spectrum`, which is explained below. For a detailed usage example, see §4.2.2.

4.2.1 Pseudo-code for `MAKE_SPECTRUM.PRO`

Items in square boxes in Figure 4.1 represent required input variables; those in rounded boxes are derived quantities. Arrows denote the dependencies. Listed below are the major steps in the algorithm and the `PINTofALE` routines that are relevant at these steps.

1. `initale,PoA_constructor :` Initialize the stellar coronal parameters electron density n_e, abundances $A(Z)$, and the absorption column density N_H (the absorption column is a function of location $[d, l_{\rm II}, b_{\rm II}]$).
2. `mk_dem(),demacs() :` Set the Differential Emission Measure distribution $DEM(T)$.
3. `rd_line(),rd_cont() :` Load the atomic line and continuum emissivities $\epsilon_l(\lambda, Z, I; T, n_e)$ and $\epsilon_c(\lambda, Z; T, n_e)$.
4. `fold_ioneq() :` Line emissivities have the option of using different ion balance calculations.
5. `lineflx() :` Compute contribution functions by folding $A(Z)$ into $\epsilon_l(T)$ and $\epsilon_c(T)$, and line and continuum intensities by integrating them, suitably weighted by $DEM(T)$.

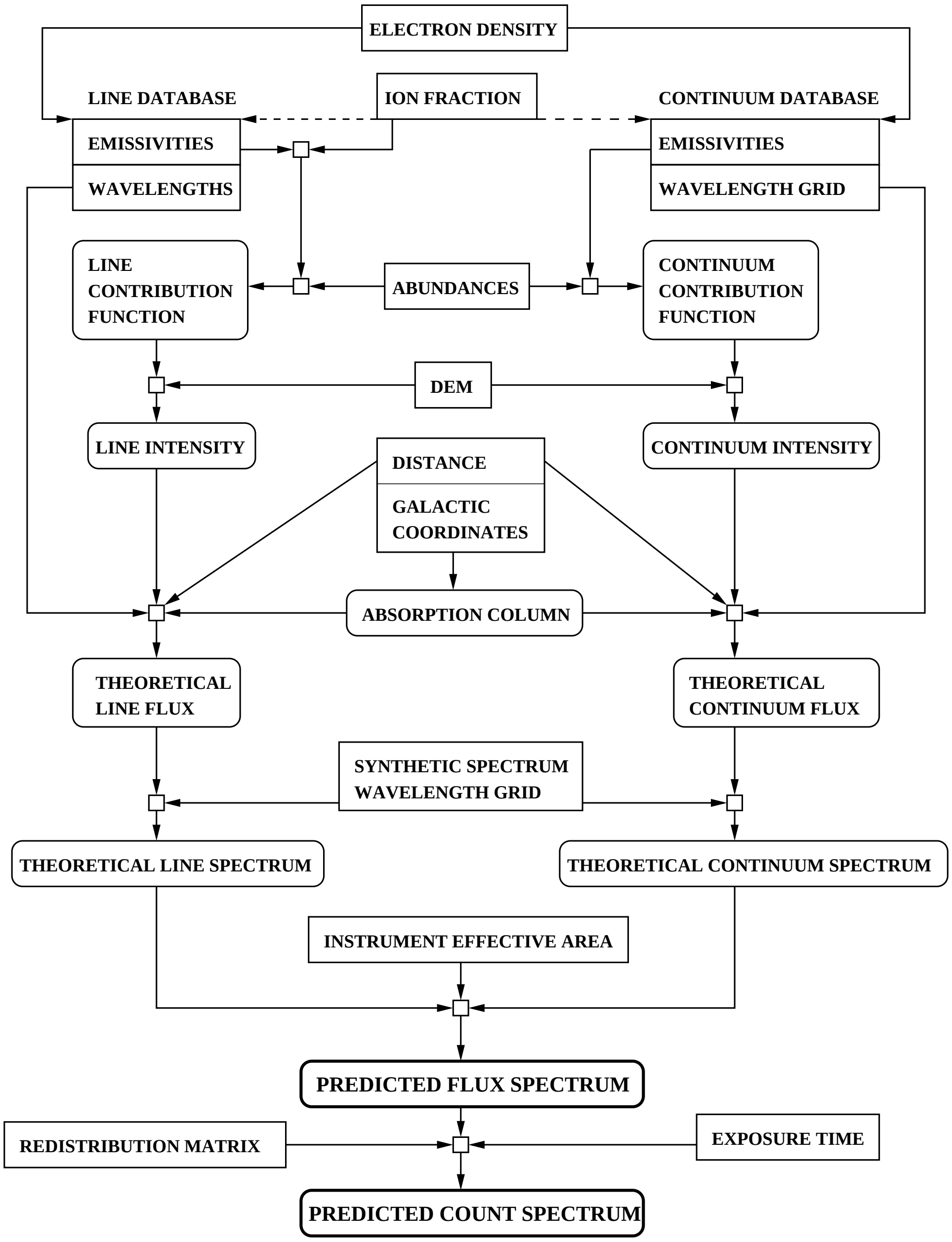

Figure 4.1: Flow chart representing the computation of a synthetic spectrum as is done, e.g., in the script make_spectrum.pro.

6. `ismabs()` : Correct for interstellar absorption and distance to obtain theoretical line and continuum fluxes at Earth.
7. `rdarf()` : Read in the effective area and define the synthetic spectrum grid (this is usually the same as the effective area wavelength grid).
8. `hastogram(),rebinw()` : Bin the line fluxes and regrid the continuum fluxes.
9. Add line and continuum spectra, weighted by effective area.
10. `conv_rmf` : Convolve the predicted flux spectrum with the response matrix and multiply by the exposure time to get the predicted count spectrum, which may then be compared with the observed spectrum

4.2.2 Generating Simulated *Chandra* ACIS X-ray Spectra

```
                   Simulation for Chandra ACIS-I (AO-9)

                                                        Last modified:
     -------------------------------------------------------------------

   This thread uses PINTofALE to generate a simulated Chandra spectrum.
   The Chandra ARF and RMF used were obtained from the Chandra Proposal
   Information page and placed in !ARDB. Here, we will generate an ACIS-I
   spectrum for a star that has a nominal ROSAT/PSPC count-rate of 0.1
   ct/s (you must have PIMMS installed locally to do this). The steps
   involved are:
    1. define a DEM
    2. rescale to match previous observation (e.g., ROSAT/PSPC)
    3. make a simulated ACIS-I spectrum

   These steps are also repeated in the associated .par file in the !ARDB
   directory, chandra_acis_ao9.par. We also show below that the spectrum
   thus generated using PINTofALE compares well with MARX simulations.
     -------------------------------------------------------------------

0. Initialize the settings

#       start IDL
idl

;       set up the PoA environment ;
.run initale
```

```
;NOTE:
;       if INITALE fails, run the script
;       @PoA_constructor

;       Change some of the preset defaults to values reasonable
;       for this exercize
!LDBDIR='$SPEX'
        ;Warning: APED emissivities include ion balance from
        ;Mazzotta et al., and abundances from Anders & Grevesse.
        ;Care must be taken to not include them twice:
        if strpos(strlowcase(!LDBDIR),'aped',0) ge 0 then !IONEQF='none'
        if strpos(strlowcase(!LDBDIR),'aped',0) ge 0 then $
                !ABUND = !ABUND/getabund('anders & grevesse')
!NH=3e20
     ------------------------------------------------------------------

1. Define a DEM

;       A Differential Emission Measure (DEM) is required to estimate
;       the amount of emission at any given temperature.  Typically
;       a 2-temperature model is used, as in the example below:
!DEM = mk_dem('delta', logT=!LOGT, pardem=alog10([7.55e6, 23.4e6]),$
        indem=[5.1d12,6.1d12]*0.05)

;       NOTE: the above is the only option available in the
;       chandra_acis_ao7.par file.

;       In general, stellar coronal plasmas are characterized by a
;       multi-temperature emission measure.  It is possible to
;       define a DEM from scratch interactively, using MK_DEM
;       (left click to define points along a DEM curve, and
;       right click to exit)
;
!DEM = mk_dem('cursor', logT=!LOGT, pardem=[1e10,1e14], $
  xcur=mk_dem_xcur, ycur=mk_dem_ycur)

;       A DEM thus derived may be edited in detail with DEMACS
;
!DEM = demacs(!LOGT, DEM0=!DEM, logT0=!LOGT, group=group, igroup=igroup)

;       more complicated DEMs, (e.g., splines) may also be defined:
;
in_pardem=[3.00000,4.01421,5.35229,5.49464,6.46261,7.06759,7.40923]
in_indem=[500.050,188.068,2.71397,2.16901,159.778,24.0164,0.133006]*1e11
```

```
!DEM = mk_dem('spline', logT=!LOGT, indem=in_indem, pardem=in_pardem) > 0

;       for the sake of definiteness, we will use a 2-temperature
;       delta-function emission measure distribution in the following
;       example.

T_components  = [6.9,    7.4]           ;log(T[K]) components in the EM
EM_components = [5.1d12, 6.1d12]        ;Emission Measure [cm^-3]
!DEM=mk_dem('delta',logT=!LOGT,pardem=T_components,indem=EM_components)

plot,!logT,!DEM,xtitle='log!d10!n(T [K])',ytitle='DEM [cm!u-5!n/logK]'

;NOTE:
;       this is a completely arbitrary DEM(T), and its only saving
;       grace is that it looks sensible.  It definitely would not
;       generate the correct number of ROSAT counts and must be
;       renormalized.
     -------------------------------------------------------------------

2. Rescale to match, e.g., ROSAT/PSPC

;       We will assume that a ROSAT/PSPC count rate is available,
;       and that the simulation will match this rate.  Data from
;       other missions such as ASCA, BeppoSAX, etc. can be dealt
;       with in the same manner.

;       First find and read in the ROSAT/PSPC effective area
;       you will need to know where your local PIMMS installation
;       is to do this.
;
pimmsdir = '/soft/pimms/data'   ;<<<change to your local installation
rosat_pspc_open = get_pimms_file('ROSAT','PSPC','OPEN',pdir=pimmsdir)
;
rd_pimms_file, rosat_pspc_open, pspc_effar, pspc_wvlar, /wave

;       make sure that the wavelengths are sorted in increasing order
;
ae=sort(pspc_wvlar) & pspc_wvlar=pspc_wvlar[ae] & pspc_effar=pspc_effar[ae]

;       some initializations
;
src_DIST=10.    ;place the source (say) 10 pc away

;       the DEM needed for the script MAKE_SPECTRUM (which will
;       be called below) must be the DEM defined at the source
```

```
;       (distance will be corrected for within the script).  Hence,
;       rescale the currently defined DEM (which has reasonable
;       values for a DEM "at Earth").
;
src_DCOR = 10.D^(alog10(4.*!DPI)+2.*alog10(src_DIST*3.1d18))
!DEM = !DEM * src_DCOR

;       now make a ROSAT/PSPC spectrum using MAKE_SPECTRUM,
;       which is a script and requires that its inputs be
;       in specially named variables and have the correct
;       units.  A more flexible approach is outlined in the
;       example sessions.
;
v_LOGT=!LOGT                    ;temperature grid
v_DEM=!DEM                      ;DEM in [cm^-3]
v_DIST=src_DIST                 ;distance in [pc]
v_EFFAR=pspc_effar > 0          ;effective area in [cm^2]
v_WVLAR=pspc_wvlar              ;effective area wavelength grid in [Ang]
v_WGRID=mid2bound(pspc_wvlar)   ;sorted wavelength bin boundaries in [Ang]
v_EXPTIME=50.                   ;nominal exposure time [ks] (say)
v_RMF='none'                    ;name of OGIP-compatible RMF
;
;NOTE:
;       note that convolving with an RMF is unnecessary at this stage
;       because all we need is a total count rate estimate
;
.run make_spectrum

;       get the total count-rate and normalize the DEM to the
;       "observed" rate of 0.1 ct/s
;
obs_rate = 0.1                                  ;[ct/s]
pred_rate = total(v_flxspc/v_EXPTIME/1e3)       ;[ct/s]
print,'normalization factor: ',obs_rate/pred_rate
;normalization factor:     0.0030849559
!DEM = !DEM * obs_rate/pred_rate
     ------------------------------------------------------------------

3. Make ACIS spectrum

;       first read in the Chandra ACIS-I on-axis ARF
;
acisi_effar=rdarf(!ARDB+'/acisi_aimpt_cy07.arf',arstr)

;       figure out the spectral binning required
```

```
;
elo=arstr.ELO & ehi=arstr.EHI & wlo=12.3985/ehi & whi=12.3985/elo

;       now reverse such that wavelengths are in increasing order
;
wlo=reverse(wlo) & whi=reverse(whi) & acisi_effar=reverse(acisi_effar)
wgrid=[wlo,max(whi)] & midwvl=0.5*(wgrid+wgrid[1:*])

;       set up to call MAKE_SPECTRUM again
;
v_LOGT=!LOGT                    ;temperature grid
v_DEM=!DEM                      ;DEM in [cm^-3]
v_EFFAR=acisi_effar             ;effective area in [cm^2]
v_WVLAR=midwvl                  ;effective area wavelength grid in [Ang]
v_WGRID=wgrid                   ;sorted wavelength bin boundaries in [Ang]
v_EXPTIME=50.                   ;nominal exposure time [ks]
v_RMF=!ARDB+'/acisi_aimpt_cy07.rmf'     ;name of OGIP-compatible RMF

;       and make the observed counts spectrum
;
.run make_spectrum

;       simulate an actual observation by finding Poisson deviates
;
nbin=n_elements(v_CTSPC)
sim_spec=intarr(nbin)
for i=0L,nbin-1L do $
  if v_CTSPC[i] gt 0 then sim_spec[i]=randomu(seed,poisson=v_CTSPC[i])

;       plot
;
plot,v_CHAN,v_CTSPC,xtitle='[keV]',ytitle='[ct]',/xl,xr=[0.5,7],$
  /yl,yr=[0.1,300],thick=2
oplot,v_CHAN,sim_spec,psym=10,thick=2,color=3
     -------------------------------------------------------------------

4.Compare with MARX simulation

;       First note that the above results are consistent with rough
;       PIMMS results.  For a ROSAT/PSPC count rate of 0.1 ct/s,
;       a Raymond-Smith spectrum of temperature 0.54 keV produces
;       approx 0.14 ct/s with Chandra/ACIS-I.  This is within a
;       factor of 2 of the value predicted here:
;
print,total(v_CTSPC)/v_EXPTIME/1e3
```

```
;        0.174697957

;       MARX requires a spectrum in an ASCII file of 2 columns,
;       with column 1 containing the energy grid in [keV] and
;       column 2 containing the flux density in [ph/s/cm^2/keV]

;       Now, v_LINSPC[v_WVLS] and v_CONSPC[v_WVLS] are already
;       almost in this form.  They must be converted from
;       [ph/s/cm^2/bin] to [ph/s/cm^2/keV].  A point to note is
;       that v_WVLS was derived within MAKE_SPECTRUM as the
;       mid-bin value of the bin boundaries in v_WGRID.

;       convert from [Ang] to [keV]
;
v_ENERGY=12.3985/v_WVLS
Egrid=12.3985/v_WGRID

;       and get the bin width
;
dE=abs(Egrid[1:*]-Egrid)

;       derive MARX compatible flux density
;
spec_MARX = (v_LINSPC + v_CONSPC) / dE

;       reverse the arrays (so that they are in increasing order of energy)
;
spec_MARX = reverse(spec_MARX)
v_ENERGY  = reverse(v_ENERGY)

;       and print out to file
;
openw,umarx,'spec_marx.dat',/get_lun
for i=0L,n_elements(v_ENERGY)-1L do printf,umarx,v_ENERGY[i],spec_MARX[i]
close,umarx & free_lun,umarx

;       see the MARX manual at http://space.mit.edu/ASC/MARX/
;       for details on how to obtain and run MARX.  Note that updating
;       to MARX 4.2.0 will allow for for Cycle 7 simulations including ACIS
;       contamination (this will be necessary to match the ARF used in this
;       example).
;
;       We have run MARX independently on the spectrum written out
;       above, using the recipe prescribed in section 5 of the manual.
;
```

```
;       cp /soft/marx/marx.par .
;       pset marx.par ExposureTime=50000 OutputDir='sim_marx' \
;         GratingType='NONE' DetectorType='ACIS-I' SourceFlux=-1 \
;         SpectrumType='FILE' SpectrumFile='spec_marx.dat' \
;         SourceType='POINT' DitherModel='INTERNAL'
;       marx TSTART=2006.46
;       marx2fits sim_marx/ spec_marx.evt
;       dmextract infile="spec_marx.evt[EVENTS][(x,y)=circle(4018,4141,10)]
;         [bin pi=1:1024:1]" outfile="spec_marx.pi" clobber=yes verbose=2

;       now read in the binned spectrum from the MARX simulation
M_CTSPC=mrdfits('spec_marx.pi',1,hdr)

;       and overplot (in red) on the PINTofALE simulation (green)
;       (note that we do not expect perfect agreement because we have
;       used a very rough ARF in section 3 above.  nevertheless, the
;       agreement is good.)
;
plot,v_CHAN,v_CTSPC,/xl,xr=[0.2,7],/xs,/yl,yr=[0.1,400],/ys,thick=2,$
  xtitle='[keV]',ytitle='[ct]',title='Comparing MARX and PINTofALE'
oplot,v_CHAN,sim_spec,psym=10,color=3
oplot,v_CHAN,M_CTSPC.COUNTS,psym=10,thick=2,color=2
```

plot made for AO-7

---

pintofale()head.cfa.harvard.edu

4.2.3 Generating Simulated *Chandra* HETG Spectra

Simulation for Chandra HETGS (AO-9)

Last modified:

This thread uses PINTofALE to generate a simulated Chandra High-Energy Transmission Grating Spectrometer (HETGS) spectrum, and in particular, we will examine the O VII density-sensitive triplet as seen in the Medium Energy Grating (MEG). The Chandra ARF and RMF used were obtained from the Chandra Proposal Information page and are assumed to be placed in the current working directory.

1. Initialize variables
2. Define a DEM

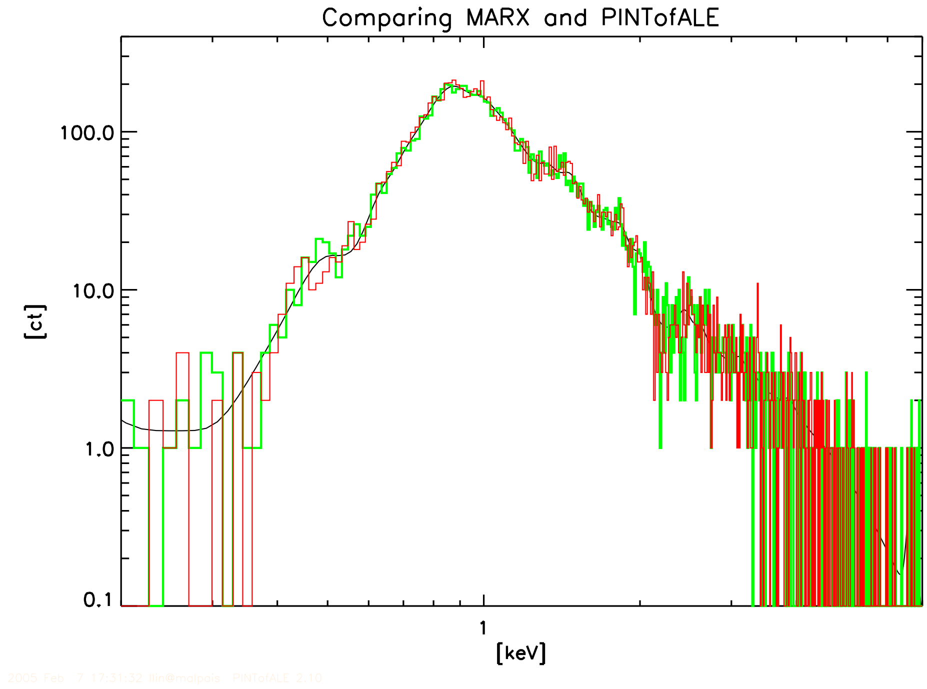

Figure 4.2: MARX simulation of `PINTofALE` spectrum predicted for ACIS-I. A flux density spectrum generated with `PINTofALE` was input to MARX and the resulting extracted spectrum is shown (thick stepped line, in red). No pileup corrections were made. The original flux density spectrum folded internally in `PINTofALE` with a nominal ACIS-I ARF and RMF (thin solid line) as well as a Poisson realization of it are also shown (thin stepped line, in green).

```
    3. Rescale DEM to match ROSAT count rate
    4. Construct spectra
    5. Simulate and plot

   All the steps described here are repeated in the .par file:
   chandra_hetg_ao9.par.
     ------------------------------------------------------------------

0. Initialize the settings

#       start IDL
idl
```

```
;       set up the PoA environment
;
.run initale

;NOTE:
;       if INITALE fails, run the script
;       @PoA_constructor
     -----------------------------------------------------------------

1. Initialize some variables

;       Because some variables will be used repeatedly in the course of this
;       thread, it might be useful to initiate them now. Nailing down these
;       variables now will also allow for something of a checklist for some
;       important quantities/files needed for the task at hand:

;       Source We will characterize our model source by the following:
 !NH           = 3e20                    ; H column density [cm^-2]
 EDENS1        = 1e09                    ; electron number density [cm^-3]
 EDENS2        = 1e12                    ; electron number density [cm^-3]
 !ABUND        = getabund('grevesse')    ; elemental abundances (SEE: getabund()
)
 T_components  = [6.3, 6.4, 7.4]         ; log(T[K]) components in the EM
 EM_components = [6.1, 6.1, 7.1]*1d12    ; Emission Measure [cm^-3]

;       Observation The observation can be described by:

 EXPTIME       = 50.                     ; nominal exposure time [ks]
 file_GRMF     = 'aciss_meg1_cy09.grmf'  ; full path name to grating RMF
 file_ARFp     = 'aciss_meg1_cy09.garf'  ; full path name to +ve order grating A
RF
 file_ARFn     = 'aciss_meg-1_cy09.garf' ; full path name to -ve order grating A
RF

;       Atomic Database  We have '$CHIANTI','$SPEX','$APED' to choose from:

 !LDBDIR = '$SPEX'

;       Analysis Parameters Theoretical wavelengths for the OVII
;       triplet (APED) are: 21.601(r),21.804(i), and 22.098(f), so to
;       maximize our efficiency we can restrict the wavelength range:

 !WMIN         = 21.5           ; minimum wavelength for output spectrum [ang]
 !WMAX         = 22.4           ; maximum wavelength for output spectrum [ang]
```

```
 nwbin         = 400L          ; number of bins in theoretical spectrum
     -----------------------------------------------------------------

2. Define a DEM

;       A Differential Emission Measure (DEM) is required to estimate the
;       amount of emission at various temperatures. Typically, a 2-temperature
;       model is used. Here we will use PINTofALE's mk_dem(), which constructs
;       a DEM array given a temperature grid and emission measure components.
;       We use as the temperature grid !LOGT. The emission measure components
;       are T_components and EM_components as defined above. We choose 'delta'
;       to treat the EM components as delta functions (see mk_dem() or the
;       Chandra/ACIS example for more options)

!DEM=mk_dem('delta', logT = !LOGT, pardem=T_components, indem=EM_components)

plot,!LOGT,!DEM,xtitle='log!d10!n(T [K])',ytitle='DEM [cm!u-5!n/logK]'

     -----------------------------------------------------------------

3. Rescale to DEM to match ROSAT count rate


;       We will assume that a ROSAT/PSPC count rate is available,
;       and that the simulation will match this rate.  We will assume
;       a count rate of 0.2 cts/s .  Data from other missions such
;       as ASCA, BeppoSAX, etc. can be dealt with in the same manner.

;       First find and read in the ROSAT/PSPC effective area.
;       You will need to know where your local PIMMS installation
;       is to do this.

pimmsdir='/soft/pimms/data'
rosat_pspc_open=get_pimms_file('ROSAT','PSPC','OPEN',pdir=pimmsdir)
rd_pimms_file, rosat_pspc_open, pspc_effar, pspc_wvlar, /wave

;       Make sure that the wavelengths are sorted in increasing order

ae=sort(pspc_wvlar) & pspc_wvlar=pspc_wvlar[ae] & pspc_effar=pspc_effar[ae]

;       The following is similar to the process described in the detailed
;       example thread (see Section 1 and Section 2).
;       For a scripted example with less details, see the companion thread
;       on the low-resolution spectrum generation.
        -----------------------------------------------------------------------
```

```
A] Read in line cooling emissivities and calculate line intensities

;Read line cooling emissivities of all possible
;lines in the ROSAT/PSPC wavelength range from the atomic data base.

!EDENS=EDENS1
lconf=rd_line(atom,n_e=!EDENS,wrange=[MIN(pspc_wvlar),MAX(pspc_wvlar)],$
              dbdir=!LDBDIR,verbose=!VERBOSE,wvl=LWVL,logT=LLOGT,Z=Z,$
              ion=ION,jon=JON,fstr=lstr)


;The output of rd_line() will only include level population,
;and not ion balances. We will use fold_ioneq() to fold ion balances.
;NOTE: This step should not be performed if !LDBDIR is set to APED,
;which already includes ion balances and abundances.

if strpos(strlowcase(!LDBDIR),'aped',0) lt 0 then lconf=$
   fold_ioneq(lconf,Z,JON,chidir=!CHIDIR,$
   logT=LLOGT,eqfile=!IONEQF,verbose=!VERBOSE)

;NOTE: If !LDBDIR is set to APED, Anders & Grevesse abundances
;are already included in the emissivities.  In such cases, either
;leave out the atomic numbers (Z) in the call to LINEFLX() below,
;or redfine the abundance array to be relative to the APED values,
;e.g.,

if strpos(strlowcase(!LDBDIR),'aped',0) ge 0 then $
   !ABUND = !ABUND/getabund('anders & grevesse')

;And now calculate line intensities using lineflx().

linint=lineflx(lconf,!LOGT,LWVL,Z,DEM=!DEM,abund=!ABUND) ;[ph/s]
-------------------------------------------------------------------------

B] Read in continuum emissivities and calculate continuum intensities

;We can read in continuum emissivities using rd_cont().
;It is important to note that the output emissivities of rd_cont()
;are in [1e-23 erg cm^3/s/Ang] and not [1e-23 erg cm^3/s] as with rd_lin
e()

!EDENS=EDENS1
cconf=rd_cont(!CEROOT,n_e=!EDENS,wrange=[min(pspc_wvlar),max(pspc_wvlar)
],$
```

```
              dbdir=!CDBDIR,abund=!ABUND,verbose=!VERBOSE,$
              wvl=CWW,logT=ClogT)

;The continuum intensities per angstrom can be calculated again using
;lineflx(). Note that CWW contains the wavelength bin boundaries for
;the emissivity array.

CWVL=0.5*(CWW[1:*]+CWW)
conint=lineflx(cconf,!LOGT,CWVL,DEM=!DEM)   ;[ph/s/Ang]

;Now to get just continuum intensity, we must multiply by an array
;containing the bin widths. If we define this array simply
;with:  CDW=CWW[1:*]-CWW, we will get an ugly 'saw-toothed' figure.
;(a side-effect of the way the data-base is constructed) To work
;around this, we can use CWVL, the mid-bin values, and mid2bound(),
;which gives intelligent bin-boundary values given mid-bin values:

CWB=mid2bound(CWVL) & CDW=CWB[1:*]-CWB
conint=conint*CDW       ;[ph/s/Ang]*[Ang]
------------------------------------------------------------------------

C] Correct for inter-stellar absorption

;Derive ISM absorptions using ismtau()
ltau=ismtau(LWVL,NH=!NH,fH2=!fH2,He1=!He1,HeII=!HeII,$
            Fano=Fano,wam=wam,/bam,abund=!ABUND,verbose=!VERBOSE)
ctau=ismtau(CWW,NH=!NH,fH2=!fH2,He1=!He1,HeII=!HeII,$
            Fano=Fano,wam=wam,/bam,abund=!ABUND,verbose=!VERBOSE)
ltrans=exp(-ltau) & ctrans=exp(-ctau)

;Derive theoretical line fluxes
linflx = linint * ltrans        ;[ph/s/cm^2]

;Derive theoretical continuum fluxes
conflx = conint * ctrans        ;[ph/s/cm^2]
------------------------------------------------------------------------

D] Bin spectra and fold in effective area

;make input theoretical spectrum grid
dwvl=float((MAX(pspc_wvlar)-MIN(pspc_wvlar))/nwbin)
wgrid=findgen(nwbin+1L)*dwvl+MIN(pspc_wvlar)

;Rebin to form theoretical line spectrum using hastrogram()
linspc = hastogram(abs(LWVL),wgrid,wts=linflx)  ;[ph/s/cm^2/bin]
```

```
        ;Rebin to form theoretical continuum spectrum using rebinw()
        conspc = rebinw(conflx,CWVL,wgrid,/perbin)      ;[ph/s/cm^2/bin]

        ;Derive predicted flux spectrum.
        WVLS=0.5*(WGRID[1:*]+WGRID)
        newEffAr=(interpol(pspc_effar,pspc_wvlar,WVLS) > 0) < (max(pspc_effar))
        flxspc = (linspc + conspc) * newEffAr

        ;Derive predicted counts spectrum.
        flxspc=flxspc*EXPTIME*1e3              ;[ct/bin]
        ------------------------------------------------------------------------


;       Now get the total count rate and renormalize the DEM to the
;       observed rate of 1.0 ct/s.

obs_rate = 1.0                             ;[ct/s]
pred_rate = total(flxspc/EXPTIME/1e3)        ;[ct/s]
print,'normalization factor: ',obs_rate/pred_rate
;normalization factor:      0.024352574
!DEM = !DEM * obs_rate/pred_rate
     ------------------------------------------------------------------

4. Construct the Spectra

;       The construction of the simulated spectra is largely a
;       repetition of the steps carried out to rescale the DEM.
;       It is in fact different only in three aspects: we will use
;       Chandra's instrumental response files and account for
;       differences in plus/minus orders.  Also, we will convolve
;       the resulting spectrum with an RMF (for the DEM normalization,
;       only the total counts was important). Thridly, because we
;       seek to look at the Oxygen triplet, we can concentrate on
;       a small wavelength range, and will read from the emissivity
;       databases accordingly.  We will repeat steps A through D as
;       above, and add a step E, RMF Convolution.  Also, we will
;       simultaneously construct spectra at two different densities:
;               n[e] = 10^12 and 10^9 [cm^-3]
        ------------------------------------------------------------------------

        A] Read in line cooling emissivities and calculate line intensities

        ;first at the higher density
```

```
        !EDENS=EDENS2
        lconf_12dex=rd_line(atom,n_e=!EDENS,wrange=[MIN(pspc_wvlar),MAX(pspc_wvl
ar)],$
                        dbdir=!LDBDIR,verbose=!VERBOSE,wvl=LWVL,logT=LLOGT,Z=Z,$
                        ion=ION,jon=JON,fstr=lstr)
        if strpos(strlowcase(!LDBDIR),'aped',0) lt 0 then lconf_12dex=$
           fold_ioneq(lconf_12dex,Z,JON,chidir=!CHIDIR,logT=LLOGT,$
                        eqfile=!IONEQF,verbose=!VERBOSE)
        linint_12dex=lineflx(lconf_12dex,!LOGT,LWVL,Z,DEM=!DEM,abund=!ABUND) ;[p
h/s]

        ;now at the lower density

        !EDENS=EDENS1
        lconf_09dex=rd_line(atom,n_e=!EDENS,wrange=[MIN(pspc_wvlar),MAX(pspc_wvl
ar)],$
                        dbdir=!LDBDIR,verbose=!VERBOSE,wvl=LWVL,logT=LLOGT,Z=Z,$
                        ion=ION,jon=JON,fstr=lstr)
         if strpos(strlowcase(!LDBDIR),'aped',0) lt 0 then lconf_09dex=$
           fold_ioneq(lconf_09dex,Z,JON,chidir=!CHIDIR,logT=LLOGT,$
                        eqfile=!IONEQF,verbose=!VERBOSE)
        linint_09dex=lineflx(lconf_09dex,!LOGT,LWVL,Z,DEM=!DEM,abund=!ABUND) ;[p
h/s]
        ------------------------------------------------------------------------

        B] Read in continuum emissivities and calculate continuum intensities

        ;at the higher density

        !EDENS=EDENS2
        cconf_12dex=rd_cont(!CEROOT,n_e=!EDENS,wrange=[!WMIN,!WMAX],$
                        dbdir=!CDBDIR,abund=!ABUND,verbose=!VERBOSE,$
                        wvl=CWW,logT=ClogT)
        CWVL=0.5*(CWW[1:*]+CWW)
        conint_12dex=lineflx(cconf_12dex,!LOGT,CWVL,DEM=!DEM)       ;[ph/s/Ang]
        CWB=mid2bound(CWVL) & CDW=CWB[1:*]-CWB
        conint_12dex=conint*CDW ;[ph/s/Ang]*[Ang]

        ;at the lower density

        !EDENS=EDENS1
        cconf_09dex=rd_cont(!CEROOT,n_e=!EDENS,wrange=[!WMIN,!WMAX],$
                        dbdir=!CDBDIR,abund=!ABUND,verbose=!VERBOSE,$
                        wvl=CWW,logT=ClogT)
        CWVL=0.5*(CWW[1:*]+CWW)
```

```
        conint_09dex=lineflx(cconf_09dex,!LOGT,CWVL,DEM=!DEM)       ;[ph/s/Ang]
        CWB=mid2bound(CWVL) & CDW=CWB[1:*]-CWB
        conint_09dex=conint*CDW ;[ph/s/Ang]*[Ang]

        ;NOTE: there is expected to be no difference between the above two
        ;because continuum emissivities are density independent for the
        ;parameter space we are operating in
        ------------------------------------------------------------------------

        C] Correct for inter-stellar absorption

        ltau=ismtau(LWVL,NH=!NH,fH2=!fH2,He1=!He1,HeII=!HeII,$
                    Fano=Fano,wam=wam,/bam,abund=!ABUND,verbose=!VERBOSE)
        ctau=ismtau(CWW,NH=!NH,fH2=!fH2,He1=!He1,HeII=!HeII,$
                    Fano=Fano,wam=wam,/bam,abund=!ABUND,verbose=!VERBOSE)
        ltrans=exp(-ltau) & ctrans=exp(-ctau)

        linflx_12dex = linint_12dex * ltrans                    ;[ph/s/cm^2]
        conflx_12dex = conint_12dex * ctrans                    ;[ph/s/cm^2]
        linflx_09dex = linint_09dex * ltrans                    ;[ph/s/cm^2]
        conflx_09dex = conint_09dex * ctrans                    ;[ph/s/cm^2]
        ------------------------------------------------------------------------

        D] Bin spectra and fold in effective area

        dwvl=float((!WMAX-!WMIN)/nwbin) & wgrid=findgen(nwbin+1L)*dwvl+!WMIN
        linspc_12dex = hastogram(abs(LWVL),wgrid,wts=linflx_12dex) ;[ph/s/cm^2/b
in]
        conspc_12dex = rebinw(conflx_12dex,CWVL,wgrid,/perbin)     ;[ph/s/cm^2/b
in]
        linspc_09dex = hastogram(abs(LWVL),wgrid,wts=linflx_09dex) ;[ph/s/cm^2/b
in]
        conspc_09dex = rebinw(conflx_09dex,CWVL,wgrid,/perbin)     ;[ph/s/cm^2/b
in]

        ;Read in the effective areas using rdarf()

        effar_p=rdarf(file_ARFp,ARSTRp)
        effar_n=rdarf(file_ARFn,ARSTRn)
        wvlar_p=!fundae.KEVANG/(0.5*(ARSTRp.ELO+ARSTRp.EHI))
        wvlar_n=!fundae.KEVANG/(0.5*(ARSTRn.ELO+ARSTRn.EHI))

        WVLS=0.5*(WGRID[1:*]+WGRID)
        newEffArP=(interpol(EFFAR_p,WVLAR_p,WVLS) > 0) < (max(EFFAR_p))
        newEffArN=(interpol(EFFAR_N,WVLAR_n,WVLS) > 0) < (max(EFFAR_n))
```

```
        ;[ct/s/bin] (if DEM is [cm^-5]: [ct/s/cm^2/bin])
        flxspcP_12dex = (linspc_12dex + conspc_12dex) * newEffArP
        flxspcN_12dex = (linspc_12dex + conspc_12dex) * newEffArN
        flxspcP_09dex = (linspc_09dex + conspc_09dex) * newEffArP
        flxspcN_09dex = (linspc_09dex + conspc_09dex) * newEffArN

        ;Derive predicted counts spectrum for +1 -1 orders
        flxspcP_12dex=flxspcP_12dex*EXPTIME*1e3                 ;[ct/bin]
        flxspcN_12dex=flxspcN_12dex*EXPTIME*1e3                 ;[ct/bin]
        flxspcP_09dex=flxspcP_09dex*EXPTIME*1e3                 ;[ct/bin]
        flxspcN_09dex=flxspcN_09dex*EXPTIME*1e3                 ;[ct/bin]
        EGRID=!fundae.KEVANG/WGRID
        ------------------------------------------------------------------------

        E] Convolve with RMF

        conv_rmf,EGRID,flxspcP_12dex,CHANP,CTSPCP_12dex,file_gRMF,rmfstr=grmf
        conv_rmf,EGRID,flxspcN_12dex,CHANN,CTSPCN_12dex,grmf
        conv_rmf,EGRID,flxspcP_09dex,CHANP,CTSPCP_09dex,grmf
        conv_rmf,EGRID,flxspcN_09dex,CHANN,CTSPCN_09dex,grmf
        ------------------------------------------------------------------------

;       Combine the two output spectra to get a co-added first order spectrum.

CHAN_dens_12dex=CHANP                           ;[keV]
cts_dens_12dex=ctspcP_12dex+ctspcN_12dex        ;[ct/bin]
CHAN_dens_09dex=CHANP                           ;[keV]
cts_dens_09dex=ctspcP_09dex+ctspcN_09dex        ;[ct/bin]

;       We thus obtain co-added spectra at two densities.

;       Note that the output energy grid of the spectra will be by
;       default the energy grid defined by the RMF.  The spectrum
;       however will only show lines and continuum between the
;       selected wavelength ranges.

     ------------------------------------------------------------------

5. Obtain Poisson deviates and plot

;       The final step is a simulation of counts based on the spectrum
;       predicted above

nbin = n_elements(cts_dens_12dex)
```

```
CTSIM_dens_12dex=intarr(nbin) & CTSIM_dens_09dex=intarr(nbin)
for i=0L,nbin-1L do if cts_dens_12dex[i] gt 0 then $
    CTSIM_dens_12dex[i]=randomu(seed,poisson=cts_dens_12dex[i])
for i=0L,nbin-1L do if cts_dens_09dex[i] gt 0 then $
    CTSIM_dens_09dex[i]=randomu(seed,poisson=cts_dens_09dex[i])

;       Now to make a comparison plot of the OVII triplet at
;       two densities (red: n[e]=10^12 cm^-3; green: n[e]=10^9 cm^-3):

plot, !fundae.KEVANG/CHAN_dens_12dex, CTSIM_dens_12dex, /nodata,$
        title='Compare O VII triplet at two densities with ACIS-S+HETGS/MEG',$
        xtitle='['+!AA+']', ytitle='[ct]', ystyle=1, xstyle=1,$
        xrange=[!WMIN,!WMAX], yrange=[0,1.1*max(CTSIM_dens_12dex)], thick=2, psy
m=10
oplot, !fundae.KEVANG/CHAN_dens_12dex, CTSIM_dens_12dex, color=2, psym=10, thick
=2
oplot, !fundae.KEVANG/CHAN_dens_09dex, CTSIM_dens_09dex, color=3, psym=10, thick
=2
stample
```

Images generated with AO-7 RMF and ARF files

(Images generated with AO-7 RMF and ARF files)

---

pintofale()head.cfa.harvard.edu (LL)

4.2.4 Generating Simulated *XMM-Newton* Spectra

Simulation for XMM spectra (AO-4)

Last modified:

This thread uses PINTofALE to generate simulated XMM EPIC-PN, EPIC-MOS, and RGS spectra of a coronal source normalized to a ROSAT/PSPC count rate. The XMM ARF, RMF and RSP files used here can be obtained from the PROPOSAL & PLANNING TOOLS page.

1. Initialize variables
2. Define a DEM
3. Rescale DEM to match ROSAT count rate
4. Construct spectra
5. Simulate and plot

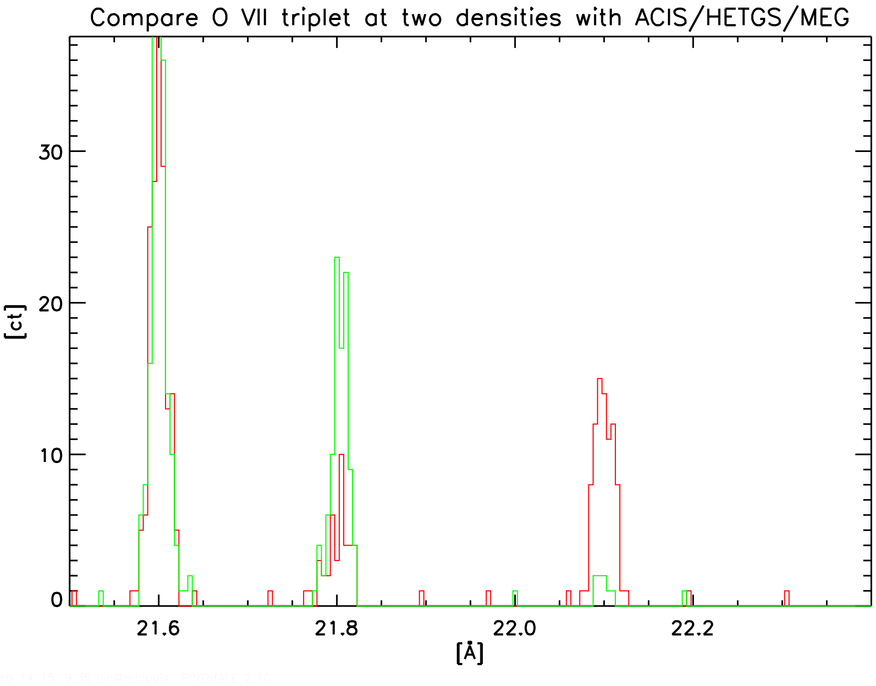

Figure 4.3: ACIS-S/HETGS/MEG spectrum of the density sensitive O VII region computed with `PINTofALE`. The predicted counts from the resonance, intercombination, and forbidden lines are shown for low-density (green; $n_e = 10^9$ cm^{-3}) and high-density (red; $n_e = 10^{12}$ cm^{-3}) cases. Nominal grating RMFs and ARFs were used to produce this realization of a coadded +ve order MEG spectrum.

```
    6. Comparison with PIMMS

   All the steps described here are repeated in the .par file:
   XMM_epic_rgs_ao4.par.
     -------------------------------------------------------------------

0. Initialize the settings

#       start IDL
idl

;       set up the PoA environment
;
```

```
.run initale

;NOTE:
;       if INITALE fails, run the script
;       @PoA_constructor
     ------------------------------------------------------------------

1. Initialize some variables

;       Because some variables will be used repeatedly in the course of this
;       thread, it might be useful to initialize them now. Nailing down these
;       variables now will also allow for something of a checklist for some
;       important quantities/files needed for the task at hand:

;       Local Environment We will set up the pathnames specific to
;       the local installation here

 !LDBDIR = '$CHIANTI'           ; Atomic Line Database
                ; choose from the predefined '$CHIANTI', '$SPEX', '$APED',
                ; or specify the full path name to the line database

 pimmsdir='/soft/pimms/data'    ; the full path name to the PIMMS database

;       Source We will characterize our model source by the following:

 !NH           = 3e20                     ; H column density [cm^-2]
 !EDENS        = 1.0e9                    ; electron number density [cm^-3]
 !ABUND        = getabund('grevesse et al.') ; element abundances
                                              ;(SEE: getabund())
 T_components  = [6.1, 6.8, 7.2]                ; log(T[K]) components in EM
 EM_components = [6.1d11, 6.1d11, 7.1e11]       ; Emission Measure [cm^-3]

;       Observation The observation can be described by:

 EXPTIME       =  50.                    ; nominal exposure time [ks]
 obs_rate      = 0.1                     ; ROSAT/PSPC count rate [ct/s]
                                         ; (set to 0 or less to be ignored)

 pn_ARF        = 'pn-medium.arf'         ; EPIC-PN ARF filename or 'none'
 mos_ARF       = 'mos1-medium.arf'       ; EPIC-MOS ARF filename or 'none'
 rgs1_ARF      = 'none'                  ; RGS1 ARF filename or 'none'
 rgs2_ARF      = 'none'                  ; RGS2 ARF filename or 'none'
 pn_RMF        = 'pn.rmf'                ; EPIC-PN RMF filename
 mos_RMF       = 'mos1.rmf'              ; EPIC-MOS RMF filename
 rgs1_RMF      = 'RGS1ORDER1.RSP'        ; RGS-1 RMF filename
```

```
 rgs2_RMF      = 'RGS2ORDER1.RSP'       ; RGS-2 RMF filename

;NOTE: the standard RGS response matrices include the
;effective area, and hence an ARF should not be specified for them.

;       Analysis Parameters
;       We may restrict the analysis to the wavelength ranges of interest.
;       Here we set the ROSAT/PSPC passband used to measure obs_rate,
;       the desired XMM output passband, and the wavelength grid
;       for the idealized model spectrum. The output spectra will be
;       defined on the default response energy grids restricted to
;       the ranges defined here.

 emin_rosat = 0.12      ; minimum energy in ROSAT/PSPC passband [keV]
 emax_rosat = 2.48      ; maximum energy in ROSAT/PSPC passband [keV]
 emin_xmm = 0.2         ; minimum energy in XMM passband [keV]
 emax_xmm = 8.0         ; maximum energy in XMM passband [keV]
 wmin_rosat = !fundae.kevang/emax_rosat ; convert to [Ang]
 wmax_rosat = !fundae.kevang/emin_rosat ; convert to [Ang]
 wmin_xmm = !fundae.kevang/emax_xmm ; convert to [Ang]
 wmax_xmm = !fundae.kevang/emin_xmm ; convert to [Ang]
 !WMIN = wmin_rosat < wmin_xmm ; minimum wavelength for model [Ang]
 !WMAX = wmax_rosat > wmax_xmm ; maximum wavelength for model [Ang]
 nwbin   =  10000L ; number of bins in model spectrum
     ------------------------------------------------------------------
```

2. Define a DEM

```
;       A Differential Emission Measure (DEM) is required to estimate the
;       amount of emission at various temperatures. Typically, a 2-temperature
;       model is used. Here we will use PINTofALE's mk_dem(), which constructs
;       a DEM array given a temperature grid and emission measure components.
;       We use as the temperature grid !LOGT. The emission measure components
;       are T_components and EM_components as defined above. We choose 'delta'
;       to treat the EM components as delta functions (see mk_dem() or the
;       Chandra/ACIS example for more options)

!DEM=mk_dem('delta', logT = !LOGT, pardem=T_components, indem=EM_components)

     ------------------------------------------------------------------
```

3. Rescale the DEM to match ROSAT count rate

```
;       We will assume that a ROSAT/PSPC count rate is available,
```

```
;       and that the simulation will match this rate.  We will assume
;       a count rate of 0.1 cts/s .  Data from other missions such
;       as ASCA, BeppoSAX, etc. can be dealt with in the same manner.

;       First find and read in the ROSAT/PSPC effective area.
;       You will need to know where your local PIMMS installation
;       is to do this.

rosat_pspc_open=get_pimms_file('ROSAT','PSPC','OPEN',pdir=pimmsdir)
rd_pimms_file, rosat_pspc_open, pspc_effar, pspc_wvlar, /wave

;       Make sure that the wavelengths are sorted in increasing order

ae=sort(pspc_wvlar) & pspc_wvlar=pspc_wvlar[ae] & pspc_effar=pspc_effar[ae]

;       The following is similar to the process described in the detailed
;       example thread (see Section 1 and Section 2)
        ------------------------------------------------------------------------

A] Read in line cooling emissivities and calculate line intensities

;       Read line cooling emissivities of all possible lines in the
;       ROSAT/PSPC wavelength range from the atomic data base.

;       NOTE: To avoid multiple reads of the line emissivity database, we
;       shall read in the emissivities over the entire range of interest

lconf=rd_line(atom,n_e=!EDENS,$
        wrange=[MIN(pspc_wvlar)<!WMIN,MAX(pspc_wvlar)>!WMAX],$
        dbdir=!LDBDIR,verbose=!VERBOSE,wvl=LWVL,logT=LLOGT,Z=Z,$
        ion=ION,jon=JON,fstr=lstr)

;       The output of rd_line() will only include level population,
;       and not ion balances. We will use fold_ioneq() to fold ion balances.
;       NOTE: This step should not be performed if !LDBDIR is set to APED,
;       which already includes ion balances and abundances.

if strpos(strlowcase(!LDBDIR),'aped',0) lt 0 then lconf=$
        fold_ioneq(lconf,Z,JON,chidir=!CHIDIR,$
        logT=LLOGT,eqfile=!IONEQF,verbose=!VERBOSE)

;       And now calculate line intensities using lineflx().

v_ABUND = !ABUND
;       NOTE: If !LDBDIR is set to APED, Anders & Grevesse abundances
```

```
;       are already included in the emissivities.  In such cases, either
;       leave out the atomic numbers (Z) in the call to LINEFLX() below,
;       or redfine the abundance array to be relative to the APED values,
;       e.g.,
if strpos(strlowcase(!LDBDIR),'aped',0) ge 0 then $
        v_ABUND = !ABUND/getabund('anders & grevesse')

linint=lineflx(lconf,!LOGT,LWVL,Z,DEM=!DEM,abund=v_ABUND) ;[ph/s]
        ------------------------------------------------------------------------

B] Read in continuum emissivities and calculate continuum intensities

;       We can read in continuum emissivities using rd_cont().
;       It is important to note that the output emissivities of rd_cont()
;       are in [1e-23 erg cm^3/s/Ang] and not [1e-23 erg cm^3/s] as with rd_line
()

;       NOTE: To avoid multiple reads of the continuum emissivity database,
;       we shall read in the emissivities over the entire range of interest

cconf=rd_cont(!CEROOT,n_e=!EDENS,$
        wrange=[min(pspc_wvlar)<!WMIN,max(pspc_wvlar)>!WMAX],$
        dbdir=!CDBDIR,abund=!ABUND,verbose=!VERBOSE,$
        wvl=CWW,logT=ClogT)

;       The continuum intensities per angstrom can be calculated again using
;       lineflx(). Note that CWW contains the wavelength bin boundaries for
;       the emissivity array.

CWVL=0.5*(CWW[1:*]+CWW)
conint=lineflx(cconf,!LOGT,CWVL,DEM=!DEM)   ;[ph/s/Ang]

;       Now to get just continuum intensity, we must multiply by an array
;       containing the bin widths. If we define this array simply
;       with:  CDW=CWW[1:*]-CWW, we will get an ugly 'saw-toothed' figure.
;       (a side-effect of the way the data-base is constructed) To work
;       around this, we can use CWVL, the mid-bin values, and mid2bound(),
;       which gives intelligent bin-boundary values given mid-bin values:

CWB=mid2bound(CWVL) & CDW=CWB[1:*]-CWB
conint=conint*CDW       ;[ph/s/Ang]*[Ang]
        ------------------------------------------------------------------------

C] Correct for inter-stellar absorption
```

```
;       Derive ISM absorptions using ismtau()
ltau=ismtau(LWVL,NH=!NH,fH2=!fH2,He1=!He1,HeII=!HeII,$
        Fano=Fano,wam=wam,/bam,abund=!ABUND,verbose=!VERBOSE)
ctau=ismtau(CWVL,NH=!NH,fH2=!fH2,He1=!He1,HeII=!HeII,$
        Fano=Fano,wam=wam,/bam,abund=!ABUND,verbose=!VERBOSE)
ltrans=exp(-ltau) & ctrans=exp(-ctau)

;       Derive theoretical line fluxes
linflx = linint * ltrans        ;[ph/s/cm^2]

;       Derive theoretical continuum fluxes
conflx = conint * ctrans        ;[ph/s/cm^2]
        ------------------------------------------------------------------------

D] Bin spectra and fold in effective area

;       make input theoretical spectrum grid
nwbin_pspc = n_elements(pspc_effar)
dwvl=float((max(pspc_wvlar)-min(pspc_wvlar))/nwbin_pspc)
wgrid=findgen(nwbin+1L)*dwvl+min(pspc_wvlar)

;       Rebin to form theoretical line spectrum using hastrogram()
linspc = hastogram(abs(LWVL),wgrid,wts=linflx)  ;[ph/s/cm^2/bin]

;       Rebin to form theoretical continuum spectrum using rebinw()
conspc = rebinw(conflx,CWVL,wgrid,/perbin)      ;[ph/s/cm^2/bin]

;       Derive predicted flux spectrum.
WVLS=0.5*(WGRID[1:*]+WGRID)
newEffAr=(interpol(pspc_effar,pspc_wvlar,WVLS) > 0) < (max(pspc_effar))
flxspc = (linspc + conspc) * newEffAr

;       Derive predicted counts spectrum.
flxspc=flxspc*EXPTIME*1e3               ;[ct/bin]

;       Restrict the counts spectrum to the specified ACIS-I range
oo = where(wgrid ge wmin_rosat and wgrid le wmax_rosat)
flxspc = flxspc(oo)
        ------------------------------------------------------------------------


;       Now get the total count rate and renormalize the DEM to the
;       observed rate of 0.1 ct/s.

pred_rate = total(flxspc/EXPTIME/1e3)       ;[ct/s]
```

```
print,''
if obs_rate gt 0 then $
        print,'Rescaling input DEM by a factor '+strtrim(obs_rate/pred_rate,2)
print,''
rescale_factor=1.0
if obs_rate gt 0 then rescale_factor = obs_rate/pred_rate
!DEM = !DEM * rescale_factor
linint = linint * rescale_factor
conint = conint * rescale_factor
linflx = linflx * rescale_factor
conflx = conflx * rescale_factor
     ------------------------------------------------------------------
```

4. Construct XMM Spectra

```
;       To construct the XMM spectra, we use the line and continuum
;       emissivities read in above (LCONF and CCONF), the line and
;       continuum intensities (LININT and CONINT) and fluxes (LINFLX
;       and CONFLX) computed above, and recompute the predicted fluxes
;       using XMM ARFs, and finally compute the observed spectra by
;       convolving with the RMFs.
        ------------------------------------------------------------------------
```

A] Read in line cooling emissivities and calculate line intensities

```
        help,lconf,linint
        ------------------------------------------------------------------------
```

B] Read in line cooling emissivities and calculate line intensities

```
        help,cconf,conint
        ------------------------------------------------------------------------
```

C] Correct for inter-stellar absorption

```
        help,ltau,ltrans,linflx,ctau,ctrans,conflx
        ------------------------------------------------------------------------
```

D] Bin spectra and fold in effective area

```
        EMAX = !fundae.kevang/!WMIN & EMIN = !fundae.kevang/!WMAX
        dnrg   = float((EMAX-EMIN)/nwbin)     ;  bin size
        egrid  = findgen(nwbin+1L)*dnrg+EMIN  ;  bin boundaries [keV]
        emid   = 0.5*(egrid[1:*]+egrid)       ;  mid-bin values [keV]
        wvls   = !fundae.kevang/emid          ;  [ang]
```

```
        linspc  = hastogram(!fundae.kevang/abs(LWVL),egrid,wts=linflx)  ; [ph/s/
cm^2/bin]
        conspc  = rebinw(conflx,!fundae.kevang/CWB,egrid,/perbin)      ; [ph/s/c
m^2/bin]

        ;Read in the effective areas using rdarf()

        if strlowcase(pn_ARF) eq 'none' then effar_pn=0*emid+1. else $
                effar_pn  = rdarf(pn_ARF,pn_ARF_str)
        if strlowcase(mos_ARF) eq 'none' then effar_mos=0*emid+1. else $
                effar_mos = rdarf(mos_ARF,mos_ARF_str)
        if strlowcase(rgs1_ARF) eq 'none' then effar_rgs1=0*emid+1. else $
                effar_rgs1 = rdarf(rgs1_ARF,rgs1_ARF_str)
        if strlowcase(rgs2_ARF) eq 'none' then effar_rgs2=0*emid+1. else $
                effar_rgs2 = rdarf(rgs2_ARF,rgs2_ARF_str)

        ;figure out the wavelength grid for effective areas

        if n_tags(pn_ARF_str) eq 0 then nrgar_pn=emid else $
          nrgar_pn = (0.5*(pn_ARF_str.ELO +pn_ARF_str.EHI))
        if n_tags(mos_ARF_str) eq 0 then nrgar_mos=emid else $
          nrgar_mos  = (0.5*(mos_ARF_str.ELO+mos_ARF_str.EHI))
        if n_tags(rgs1_ARF_str) eq 0 then nrgar_rgs1=emid else $
          nrgar_rgs1  = (0.5*(rgs1_ARF_str.ELO+rgs1_ARF_str.EHI))
        if n_tags(rgs2_ARF_str) eq 0 then nrgar_rgs2=emid else $
          nrgar_rgs2  = (0.5*(rgs2_ARF_str.ELO+rgs2_ARF_str.EHI))

        ;interpolate to put effective area on binned spectra grids

        new_effar_pn   = (interpol(effar_pn  ,nrgar_pn  ,EMID) > 0) < (max(effar
_pn))
        new_effar_mos  = (interpol(effar_mos ,nrgar_mos ,EMID) > 0) < (max(effar
_mos))
        new_effar_rgs1 = (interpol(effar_rgs1,nrgar_rgs1,EMID) > 0) < (max(effar
_rgs1))
        new_effar_rgs2 = (interpol(effar_rgs2,nrgar_rgs2,EMID) > 0) < (max(effar
_rgs2))

        ;[ct/s/bin] (if DEM is [cm-5]: [ct/s/cm2/bin])

        flxspc_pn   = (linspc + conspc) * new_effar_pn
        flxspc_mos  = (linspc + conspc) * new_effar_mos
        flxspc_rgs1 = (linspc + conspc) * new_effar_rgs1
        flxspc_rgs2 = (linspc + conspc) * new_effar_rgs2
```

```
        ;Derive predicted counts spectrum

        flxspc_pn    = flxspc_pn   *EXPTIME*1e3      ;[ct/bin]
        flxspc_mos   = flxspc_mos  *EXPTIME*1e3      ;[ct/bin]
        flxspc_rgs1  = flxspc_rgs1 *EXPTIME*1e3      ;[ct/bin]
        flxspc_rgs2  = flxspc_rgs2 *EXPTIME*1e3      ;[ct/bin]
        ------------------------------------------------------------------------

E] Convolve with RMF using

;       read in RMFs
pn_RMF_str=rd_ogip_rmf(pn_RMF)
mos_RMF_str=rd_ogip_rmf(mos_RMF)
rgs1_RMF_str=rd_ogip_rmf(rgs1_RMF)
rgs2_RMF_str=rd_ogip_rmf(rgs2_RMF)

conv_rmf, egrid, flxspc_pn,   CHAN_pn,   CTSPC_pn,   pn_RMF_str
conv_rmf, egrid, flxspc_mos,  CHAN_mos,  CTSPC_mos,  mos_RMF_str
conv_rmf, egrid, flxspc_rgs1, CHAN_rgs1, CTSPC_rgs1, rgs1_RMF_str
conv_rmf, egrid, flxspc_rgs2, CHAN_rgs2, CTSPC_rgs2, rgs2_RMF_str
        ------------------------------------------------------------------------

;       Get co-added RGS spectrum.

CHAN_rgs   = CHAN_rgs1                        ;[keV]
CTSPC_rgs  = CTSPC_rgs1 + CTSPC_rgs2          ;[ct/bin] valid grids same
flxspc_rgs = flxspc_rgs1+flx_spc_rgs2         ;[ct/bin] (if sperate ARF)

;       Restrict the spectra to the specified range
oo_rgs = where(CHAN_rgs gt emin_xmm and CHAN_rgs lt emax_xmm)
oo_mos = where(CHAN_mos gt emin_xmm and CHAN_mos lt emax_xmm)
oo_pn  = where(CHAN_pn  gt emin_xmm and CHAN_pn lt emax_xmm)

CTSPC_rgs1 = CTSPC_rgs1(oo_rgs) & CHAN_rgs1  = CHAN_rgs1(oo_rgs)
CTSPC_rgs2 = CTSPC_rgs2(oo_rgs) & CHAN_rgs2  = CHAN_rgs2(oo_rgs)
CTSPC_rgs  = CTSPC_rgs(oo_rgs)  & CHAN_rgs   = CHAN_rgs(oo_rgs)
CTSPC_mos  = CTSPC_mos(oo_mos)  & CHAN_mos   = CHAN_mos(oo_mos)
CTSPC_pn   = CTSPC_mos(oo_pn)   & CHAN_pn    = CHAN_pn(oo_pn)

;       Note that the output energy grid of the spectra will be by
;       default the energy grid defined by the RMF. The spectrum
;       however will only show lines and continuum between the
;       selected wavelength ranges.
```

```
    -----------------------------------------------------------------

5. Obtain Poisson deviates and plot

;       The final step is a simulation of counts based on the spectrum
;       predicted above

nbin_pn  = n_elements(CTSPC_pn)  &  CTSIM_pn  = intarr(nbin_pn)
nbin_mos = n_elements(CTSPC_mos) &  CTSIM_mos  = intarr(nbin_mos)
nbin_rgs = n_elements(CTSPC_rgs) &  CTSIM_rgs = intarr(nbin_rgs)

for i=0L,nbin_pn-1L  do if CTSPC_pn[i]  gt 0 then $
        CTSIM_pn[i] =randomu(seed,poisson=CTSPC_pn[i])
for i=0L,nbin_mos-1L do if CTSPC_mos[i] gt 0 then $
        CTSIM_mos[i]=randomu(seed,poisson=CTSPC_mos[i])
for i=0L,nbin_rgs-1L do if CTSPC_rgs[i] gt 0 then $
        CTSIM_rgs[i]=randomu(seed,poisson=CTSPC_rgs[i])

;       The results of the calculations are plotted below

;       The EPIC-pn model (in white) and simulated counts (in red)
window, 0
plot,   CHAN_pn, CTSPC_pn, title='XMM EPIC-PN MODEL SPECTRUM',/xl,/yl,$
        xtitle='[keV]', ytitle='[ct]', ystyle=1, xstyle=1, $
        xrange=[emin_xmm,emax_xmm], yrange=[0.5,2*max(CTSPC_pn)] & stample
oplot,  CHAN_pn, CTSIM_pn ,color = 2, psym=10,  thick=2


;       The EPIC-mos model (in white) and simulated counts (in red)
window, 1
plot,   CHAN_mos, CTSPC_mos, title='XMM EPIC-MOS MODEL SPECTRUM',/xl,/yl,$
        xtitle='[keV]', ytitle='[ct]', ystyle=1, xstyle=1, $
        xrange=[emin_xmm,emax_xmm], yrange=[0.5,2*max(CTSPC_mos)] & stample
oplot,  CHAN_mos, CTSIM_mos, color = 2, psym=10, thick=2


;       The RGS model (in white) and simulated counts (in red).  Also shown
;       are the model counts from RGS1 (yellow) and RGS2 (green)
window, 2
plot,   !fundae.kevang/CHAN_rgs, CTSIM_rgs, title='XMM RGS MODEL SPECTRUM',$
        xtitle='['+!AA+']', ytitle='[ct]', ystyle=1, xstyle=1, /nodata,$
        xrange=[!fundae.kevang/MAX(CHAN_rgs),!fundae.kevang/MIN(CHAN_rgs)],$
        yrange=[0,1.1*max(CTSIM_rgs)]
oplot,  !fundae.kevang/CHAN_rgs, CTSIM_rgs, color=2, psym=10, thick=2
oplot,  !fundae.kevang/CHAN_rgs1, CTSPC_rgs1, color=3, psym=10, thick=2
```

```
oplot,  !fundae.kevang/CHAN_rgs2, CTSPC_rgs2, color=4, psym=10, thick=2


;       As above, but zoomed in to show the low counts detail.
window, 3
plot,   !fundae.kevang/CHAN_rgs, CTSIM_rgs, title='XMM RGS MODEL SPECTRUM',$
        xtitle='['+!AA+']', ytitle='[ct]', ystyle=1, xstyle=1, /nodata,$
        xrange=[!fundae.kevang/MAX(CHAN_rgs),!fundae.kevang/MIN(CHAN_rgs)],$
        yrange=[0,5.*median(CTSPC_rgs)>5.]
oplot,  !fundae.kevang/CHAN_rgs, CTSIM_rgs, color=2, psym=10
oplot,  !fundae.kevang/CHAN_rgs1, CTSPC_rgs1, color=3, psym=10
oplot,  !fundae.kevang/CHAN_rgs2, CTSPC_rgs2, color=4, psym=10


;       We may summarize results as follows:
;
; Simulated counts spectra are in:
;    CTSIM_pn(chan_pn),CTSIM_mos(CHAN_mos),CTSIM_rgs(CHAN_rgs) [ct/bin]
;
; Redistributed counts spectra are in:
;    CTSPC_pn(chan_pn),CTSPC_mos(CHAN_mos),CTSPC_rgs(CHAN_rgs) [ct/bin]
;
; Predicted counts spectra are in:
;    FLXSPC_pn(wvls),FLXSPC_mos(wvls), FLXSPC_rgs(wvls) [ct/bin]
;
;    Note: Standard RGS response matrices include the effective area, so, unless
;    a seperate rmf and effective area files are used, FLXSPC_rgs will
;    NOT contain predicted counts spectra.
;
; Theoretical line fluxes are in:
;    linspc(lwvl) [ph/s[/cm^2]]
;
; Theoretical continuum fluxes are in:
;    conspc(cwvl) [ph/s[/cm^2]]

     -------------------------------------------------------------------

6. Comparison with PIMMS

;       Results using this thread  are roughly consistent with
;       PIMMS results. PIMMS returned the following XMM count rates
;       for a ROSAT/PSPC count rate of 0.1 cts/s and a Raymond-Smith
;       model of temperature 0.5437 keV:
;             XMM PN MED  :   7.792E-01 cts/s
;             XMM MOS MED :   1.958E-01 cts/s
```

```
;            XMM RGS1 O1 :   2.629E-02 cts/s
;            XMM RGS2 O1 :   3.627E-02 cts/s
;
;      Running this thread with  !ABUND = getabund('Allen'),
;      T_components = [6.8], and  obs_rate = 0.1 gives:
;            XMM PN MED   :  0.74  cts/s
;            XMM MOS MED :  0.19  cts/s
;            XMM RGS1 O1 :  0.027 cts/s
;            XMM RGS2 O1 :  0.035 cts/s
     ------------------------------------------------------------------


   pintofale()head.cfa.harvard.edu (LL)
```

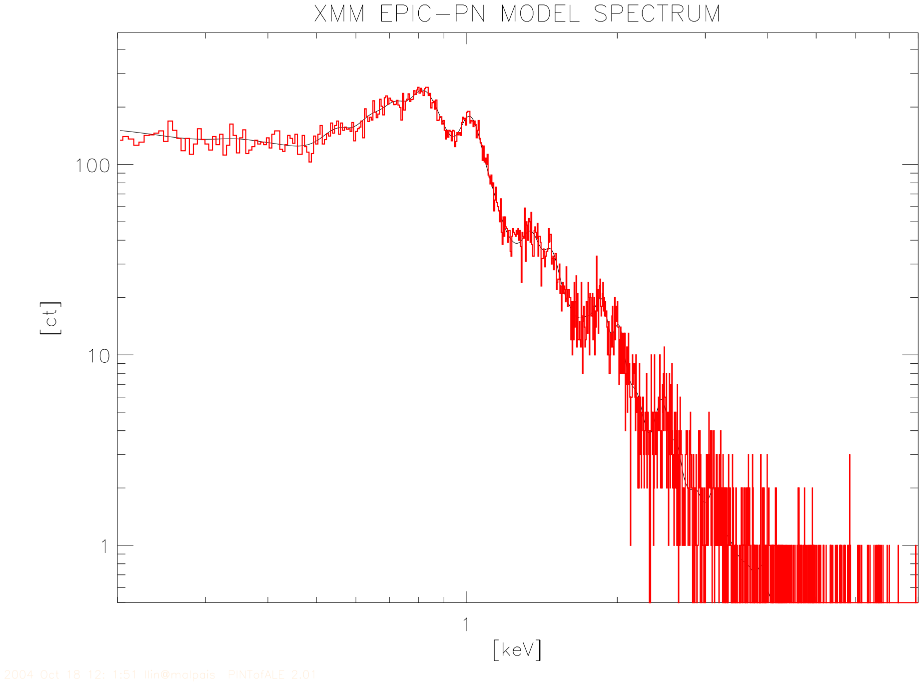

Figure 4.4: XMM/EPIC-pn spectrum computed with `PINTofALE`. The predicted model is shown as the smooth line and the simulated counts spectrum is shown in red.

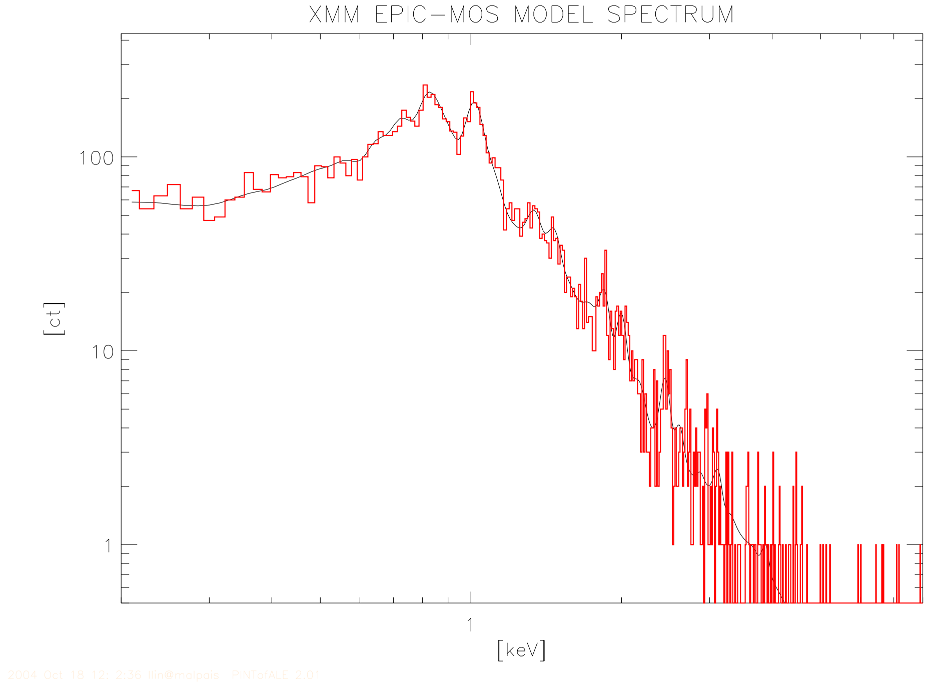

Figure 4.5: XMM/EPIC-mos spectrum computed with `PINTofALE`. The predicted model is shown as the smooth line and the simulated counts spectrum is shown in red.

4.2.5 Generating Simulated *Suzaku* XIS Spectra

```
                    Simulation for Suzaku spectra (AO-2)

                                                              Last modified:
  ------------------------------------------------------------------

This thread uses PINTofALE to generate Suzaku/XIS spectra and
Suzaku/HXD count rate estimates for a coronal source, using a specified
multi-temperature DEM, normalized to a Chandra/ACIS-I count rate. The
Suzaku ARF, RMF, and background files used here can be obtained from
heasarc.gsfc.nasa.gov/docs/astroe/prop_tools/xis_mat.html
for XIS and
heasarc.gsfc.nasa.gov/docs/astroe/prop_tools/hxd_mat.html
for HXD.

All the steps described here are repeated in the .par file:
```

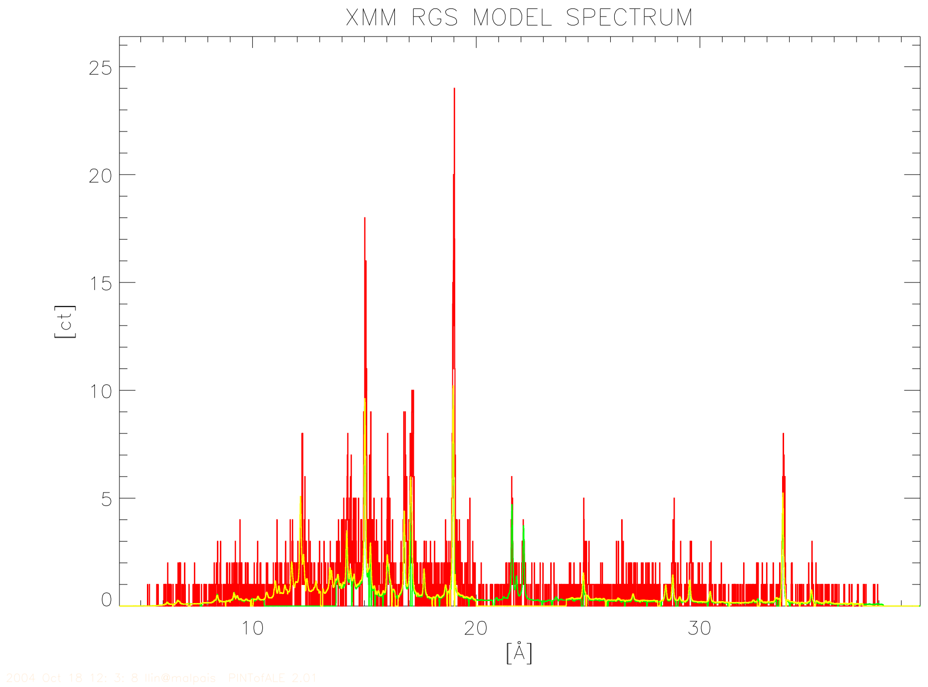

Figure 4.6: XMM/RGS1+RGS2 coadded spectrum computed with `PINTofALE`. The predicted model is shown as the smooth line and the simulated counts spectrum is shown in red. In addition, the individual contributions from each RGS are shown in yellow (RGS1) and green (RGS2)

```
   Suzaku_ao2.par.
     -----------------------------------------------------------------

Initialize the settings

#       start IDL
idl

;       set up the PoA environment
;
.run initale

;NOTE:
;       if INITALE fails, run the script
```

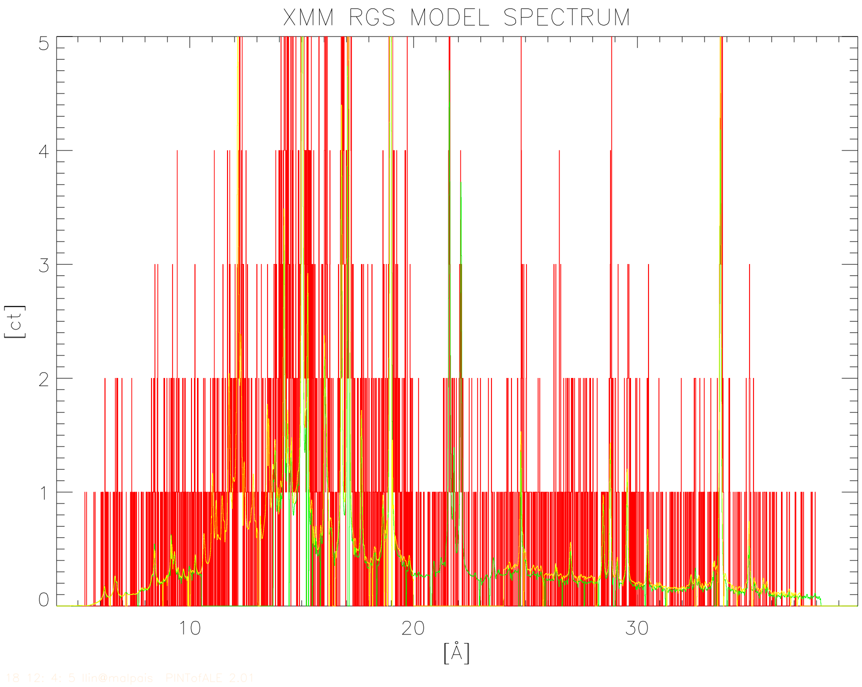

Figure 4.7: As in Figure 4.6, but zoomed in to show the details in the low-counts region.

```
;       @PoA_constructor

Set up the simulation control variables

   Set up the local environment variables, such as pathnames specific to
   the installation, source model parameters, and observation specific
   variables.
pimmsdir        = '/soft/pimms/data'    ; change to your local installation

!LDBDIR         = '$SPEX'                         ; Atomic Line Database
                ; choose from the predefined '$CHIANTI', '$SPEX', '$APED',
                ; or specify the full path name to the line database
!IONEQF         = 'ioneq/mazzotta_etal.ioneq'     ; ion balance
!CEROOT         = 'chianti'                       ; 'cie' or 'chianti'

!NH             = 3e20                            ; H column density [cm^-2]
!EDENS          = 1.0e9                           ; electron number density [cm^-3
]
```

```
!ABUND          = getabund('asplund et al.')    ; element abundances
T_components    = [6.1, 6.8, 7.2]               ; log(T[K]) components in EM
EM_components   = [6.1d11, 6.1d11, 7.1e11]      ; Emission Measure [cm^-5]

EXPTIME         = 30.                           ; nominal exposure time [ks]
acisi_rate      = 0.1   ; Chandra/ACIS-I rate [ct/s] - 0 or less to ignore
acisi_emin      = 0.3   ; minimum ACIS-I energy to consider [keV]
acisi_emax      = 8.0   ; maximum ACIS-I energy to consider [keV]
cornorm         = 0.0                           ; background correction norm

   Provide the names of the files containing the ARFs, RMFs, and
   background PHAs. Note that 'none' is an acceptable option. XIS related
   files can be obtained from
   heasarc.gsfc.nasa.gov/docs/astroe/prop_tools/xis_mat.html
   and HXD related files from
   heasarc.gsfc.nasa.gov/docs/astroe/prop_tools/hxd_mat.html.
xisFI_RMF       = 'ae_xi2_20051219.rmf'         ; XIS FI RMF filename
xisFI_ARF       = 'ae_xi2_onaxis_20050916.arf'  ; XIS FI ARF filename
xisFI_BKG       = 'ae_xi2_back.pha'             ; XIS FI bkg PHA filename
;
xisBI_RMF       = 'ae_xi1_20051210d.rmf'        ; XIS BI RMF filename
xisBI_ARF       = 'ae_xi1_onaxis_20050916.arf'  ; XIS BI ARF filename
xisBI_BKG       = 'ae_xi1_back.pha'             ; XIS BI bkg PHA filename

hxdPIN_RSP      = 'ae_hxd_pinxinom_20051104.rsp'        ;HXD PIN RSP filename
hxdPIN_BKG      = 'ae_hxd_pinbkg_20051105.pha'          ;HXD PIN bkg PHA filenam
e
;
hxdGSO_RSP      = 'ae_hxd_gso_20051019.rsp'             ;HXD GSO RSP filename
hxdGSO_BKG      = 'ae_hxd_gsobkg_20051105.pha'          ;HXD GSO bkg PHA filenam
e

   These variables will be computed or determined downstream, but this
   thread is set up to speed up calculations by not repeating unnecessary
   tasks. Unless explicitly uncommented, inputs corresponding to the
   following will not be read in again after the first run.
;  ae2_lstr     = 0     ;the line emissivity database structure
;  ae2_cstr     = 0     ;the XIS continuum emissivity database structure
;  ae2_hcstr    = 0     ;the HXD continuum emissivity database structure
;  rm_xisFI     = 0     ;the XIS/FI response matrix
;  ea_xisFI     = 0     ;the XIS/FI effective area
;  bg_xisFI     = 0     ;the XIS/FI background spectrum
;  rm_xisBI     = 0     ;the XIS/BI response matrix
;  ea_xisBI     = 0     ;the XIS/BI effective area
;  bg_xisBI     = 0     ;the XIS/BI background spectrum
```

```
;  rm_hxdPIN    = 0     ;the HXD/PIN response matrix
;  bg_hxdPIN    = 0     ;the HXD/PIN background spectrum
;  rm_hxdGSO    = 0     ;the HXD/GSO response matrix
;  bg_hxdGSO    = 0     ;the HXD/GSO background spectrum

Compute the DEM

   A Differential Emission Measure (DEM) is required to estimate the
   amount of emission at various temperatures. Typically, a 2-temperature
   model is used. Here we will use PINTofALE's mk_dem(), which constructs
   a DEM array given a temperature grid and emission measure components.
   We use as the temperature grid !LOGT. The emission measure components
   are T_components and EM_components as defined above
!DEM=mk_dem('delta', logT = !LOGT, pardem=T_components, indem=EM_components)

Read in the Suzaku calibration data

;       XIS/FI
dwvl=0.000434 & xis_FIwgrid=10.^(findgen(4097)*dwvl)
if strlowcase(xisFI_RMF) ne 'none' and n_tags(rm_xisFI) eq 0 then $
  rm_xisFI=rd_ogip_rmf(xisFI_RMF,effar=qe_xisFI)
if n_tags(rm_xisFI) ne 0 then xis_FIwgrid=12.3985/[min(rm_xisFI.ELO),rm_xisFI.EH
I]
if strlowcase(xisFI_RMF) eq 'none' and n_tags(rm_xisFI) eq 0 then qe_xisFI=0.*xi
s_FIwgrid[1:*]+1.
if strlowcase(xisFI_ARF) ne 'none' and not keyword_set(ea_xisFI) then ea_xisFI=r
darf(xisFI_ARF,ea_xisFIstr)
if strlowcase(xisFI_ARF) eq 'none' and not keyword_set(ea_xisFI) then ea_xisFI=0
.*xis_FIwgrid[1:*]+1.
if strlowcase(xisFI_BKG) ne 'none' and n_tags(bg_xisFI) eq 0 then bg_xisFI=mrdfi
ts(xisFI_BKG,1,hbg_xisFI)
if strlowcase(xisFI_BKG) eq 'none' and n_tags(bg_xisFI) eq 0 then $
  bg_xisFI=create_struct('CHANNEL',lindgen(n_elements(xis_FIwgrid)-1),'COUNTS',l
onarr(0*xis_FIwgrid[1:*]))
bg_xisFI_exp=1. & if keyword_set(hbg_xisFI) then bg_xisFI_exp=float(sxpar(hbg_xi
sFI,'EXPOSURE'))>1.

;       XIS/BI
dwvl=0.000434 & xis_BIwgrid=10.^(findgen(4097)*dwvl)
if strlowcase(xisBI_RMF) ne 'none' and n_tags(rm_xisBI) eq 0 then $
  rm_xisBI=rd_ogip_rmf(xisBI_RMF,effar=qe_xisBI)
if n_tags(rm_xisBI) ne 0 then xis_BIwgrid=12.3985/[min(rm_xisBI.ELO),rm_xisBI.EH
I]
if strlowcase(xisBI_RMF) eq 'none' and n_tags(rm_xisBI) eq 0 then qe_xisBI=0.*xi
s_BIwgrid[1:*]+1.
```

```
if strlowcase(xisBI_ARF) ne ’none’ and not keyword_set(ea_xisBI) then ea_xisBI=r
darf(xisBI_ARF,ea_xisBIstr)
if strlowcase(xisBI_ARF) eq ’none’ and not keyword_set(ea_xisBI) then ea_xisBI=0
.*xis_BIwgrid[1:*]+1.
if strlowcase(xisBI_BKG) ne ’none’ and n_tags(bg_xisBI) eq 0 then bg_xisBI=mrdfi
ts(xisBI_BKG,1,hbg_xisBI)
if strlowcase(xisBI_BKG) eq ’none’ and n_tags(bg_xisBI) eq 0 then $
  bg_xisBI=create_struct(’CHANNEL’,lindgen(n_elements(xis_BIwgrid)-1),’COUNTS’,l
onarr(0*xis_BIwgrid[1:*]))
bg_xisBI_exp=1. & if keyword_set(hbg_xisBI) then bg_xisBI_exp=float(sxpar(hbg_xi
sBI,’EXPOSURE’))>1.

;       HXD/PIN
dwvl=0.0094 & hxd_PINwgrid=10.^(findgen(257)*dwvl-0.89)
if strlowcase(hxdPIN_RSP) ne ’none’ and n_tags(rm_hxdPIN) eq 0 then $
  rm_hxdPIN=rd_ogip_rmf(hxdPIN_RSP,effar=ea_hxdPIN)
if n_tags(rm_hxdPIN) ne 0 then hxd_PINwgrid=12.3985/[min(rm_hxdPIN.ELO),rm_hxdPI
N.EHI]
if n_tags(rm_hxdPIN) ne 0 then ea_hxdPIN=total(rm_hxdPIN.MATRIX,1) else ea_hxdPI
N=0.*hxd_PINwgrid[1:*]+1.
if strlowcase(hxdPIN_BKG) ne ’none’ and n_tags(bg_hxdPIN) eq 0 then bg_hxdPIN=mr
dfits(hxdPIN_BKG,1,hbg_hxdPIN)
if strlowcase(hxdPIN_BKG) eq ’none’ and n_tags(bg_hxdPIN) eq 0 then $
  bg_hxdPIN=create_struct(’CHANNEL’,lindgen(n_elements(hxd_PINwgrid)-1),’COUNTS’
,lonarr(0*hxd_PINwgrid[1:*]))
bg_hxdPIN_exp=1. & if keyword_set(hbg_hxdPIN) then bg_hxdPIN_exp=float(sxpar(hbg
_hxdPIN,’EXPOSURE’))>1.

;       HXD/GSO
dwvl=0.0059 & hxd_GSOwgrid=10.^(findgen(513)*dwvl-1.92)
if strlowcase(hxdGSO_RSP) ne ’none’ and n_tags(rm_hxdGSO) eq 0 then $
  rm_hxdGSO=rd_ogip_rmf(hxdGSO_RSP,effar=ea_hxdGSO)
if n_tags(rm_hxdGSO) ne 0 then hxd_GSOwgrid=12.3985/[min(rm_hxdGSO.ELO),rm_hxdGS
O.EHI]
if n_tags(rm_hxdGSO) ne 0 then ea_hxdGSO=total(rm_hxdGSO.MATRIX,1) else ea_hxdGS
O=0.*hxd_GSOwgrid[1:*]+1.
if strlowcase(hxdGSO_BKG) ne ’none’ and n_tags(bg_hxdGSO) eq 0 then bg_hxdGSO=mr
dfits(hxdGSO_BKG,1,hbg_hxdGSO)
if strlowcase(hxdGSO_BKG) eq ’none’ and n_tags(bg_hxdGSO) eq 0 then $
  bg_hxdGSO=create_struct(’CHANNEL’,lindgen(n_elements(hxd_GSOwgrid)-1),’COUNTS’
,lonarr(0*hxd_GSOwgrid[1:*]))
bg_hxdGSO_exp=1. & if keyword_set(hbg_hxdGSO) then bg_hxdGSO_exp=float(sxpar(hbg
_hxdGSO,’EXPOSURE’))>1.

Read in line and continuum emissivity databases
```

```
wmin=min(xis_BIwgrid) < min(xis_FIwgrid)
wmax=max(xis_BIwgrid) < max(xis_FIwgrid)
incieq=1
if strpos(strlowcase(!LDBDIR),'apec',0) ge 0 then incieq=0 else $
 if strpos(strlowcase(!LDBDIR),'aped',0) ge 0 then incieq=0 else $
  if strpos(strlowcase(!LDBDIR),'gu',0) ge 0 then incieq=0
if n_tags(ae2_lstr) eq 0 then $
  ae2_lstr=rd_list('all|'+strtrim(wmin,2)+'-'+strtrim(wmax,2)+'|'+!LDBDIR,sep='|
',$
  incieq=incieq,n_e=!EDENS,eqfile=!IONEQF,verbose=!VERBOSE)

;       If !LDBDIR is set to APED, Anders & Grevesse abundances
;       are already included in the emissivities.  In which case,
;       redefine the abundance array to be relative to APED.

abnorm=0*!ABUND+1.
if strpos(strlowcase(!LDBDIR),'apec',0) ge 0 or $
  strpos(strlowcase(!LDBDIR),'aped',0) ge 0 then $
  abnorm=getabund('anders & grevesse')

if n_tags(ae2_cstr) eq 0 then $
  fc=rd_cont(!CEROOT,n_e=!EDENS,wrange=[wmin,wmax],dbdir=!CDBDIR,$
  abund=!ABUND,fcstr=ae2_cstr,twoph=twoph)

if n_tags(ae2_hcstr) eq 0 then $
  fc=rd_cont(!CEROOT,n_e=!EDENS,wrange=12.3985/[60.,1.],dbdir=!CDBDIR,$
  abund=!ABUND,fcstr=ae2_hcstr,twoph=twoph)

Compute nominal fluxes and ISM opacities

;       line intensities
linint= lineflx(ae2_lstr.LINE_INT,ae2_lstr.LOGT,abs(ae2_lstr.WVL),$
        ae2_lstr.Z,DEM=!DEM,abund=!ABUND/abnorm)                ;[ph/s/cm^2]

;       continuum intensities
conint= lineflx(ae2_cstr.CONT_INT,ae2_cstr.LOGT,abs(ae2_cstr.midWVL),$
        DEM=!DEM)                                               ;[ph/s/cm^2/Ang]
cwb=    mid2bound(ae2_cstr.midWVL)
cdw=    cwb[1:*]-cwb
conint= conint*cdw                                              ;[ph/s/cm^2]

;       continuum intensities for HXD
hconint=lineflx(ae2_hcstr.CONT_INT,ae2_hcstr.LOGT,abs(ae2_hcstr.midWVL),$
        DEM=!DEM)                                               ;[ph/s/cm^2/Ang]
```

```
cwb=    mid2bound(ae2_hcstr.midWVL)
cdw=    cwb[1:*]-cwb
hconint=hconint*cdw                                                 ;[ph/s/cm^2]

;       ISM absorption
ltau=   ismtau(abs(ae2_lstr.WVL),NH=!NH,fH2=!fH2,He1=!He1,HeII=!HeII,$
        Fano=Fano,wam=wam,/bam,abund=!ABUND,verbose=!VERBOSE)
ctau=   ismtau(abs(ae2_cstr.midWVL),NH=!NH,fH2=!fH2,He1=!He1,HeII=!HeII,$
        Fano=Fano,wam=wam,/bam,abund=!ABUND,verbose=!VERBOSE)
hctau=  ismtau(abs(ae2_hcstr.midWVL),NH=!NH,fH2=!fH2,He1=!He1,HeII=!HeII,$
        Fano=Fano,wam=wam,/bam,abund=!ABUND,verbose=!VERBOSE)
ltrans= exp(-ltau)
ctrans= exp(-ctau)
hctrans=exp(-hctau)

;       line fluxes
linflx= linint * ltrans         ; [ph/s/cm^2]
conflx= conint * ctrans         ; [ph/s/cm^2]
hconflx=hconint * hctrans       ; [ph/s/cm^2]
```

renormalize DEM by comparing to predicted ACIS rate

We will assume that a Chandra/ACIS-I count rate is available, and require that the simulation match this rate. Data from other missions such as ROSAT, ASCA, BeppoSAX, etc. can be dealt with in the same manner.

```
;       First find and read in the Chandra/ACIS-I effective area
;       you will need to know where your local PIMMS installation
;       is to do this.

rd_pimms_file,get_pimms_file('chandra','acis-i',pdir=pimmsdir),$
        acisi_effar,acisi_wvlar,/wave

;       fold in effective area
acisi_wmin=12.3985/acisi_emax & acisi_wmax=12.3985/acisi_emin
os=sort(acisi_wvlar) & acisi_wvlar=acisi_wvlar[os] & acisi_effar=acisi_effar[os]
;
lwvl=abs(ae2_lstr.WVL) & ow=where(lwvl ge acisi_wmin and lwvl le acisi_wmax)
effar=(interpol(acisi_effar,acisi_wvlar,lwvl[ow])>0) < max(acisi_effar)
flxlin= linflx[ow]*effar                ;[ct/s]
;
cwvl=abs(ae2_cstr.midWVL) & ow=where(lwvl ge acisi_wmin and lwvl le acisi_wmax)
effar=(interpol(acisi_effar,acisi_wvlar,cwvl)>0) < max(acisi_effar)
flxcon= conflx[ow]*effar                ;[ct/s]
```

```
;       total predicted rate in Chandra/ACIS-I
pred_rate=total(flxlin)+total(flxcon)

print,''
if acisi_rate gt 0 then $
  print,'Rescaling input DEM by a factor '+strtrim(acisi_rate/pred_rate,2)
  print,'to match required ACIS-I rate of '+strtrim(acisi_rate,2)+' ct/s'
print,''
rescale_factor=1.0
if acisi_rate gt 0 then rescale_factor = acisi_rate/pred_rate
!DEM = !DEM * rescale_factor
linflx  = linflx * rescale_factor
conflx  = conflx * rescale_factor
hconflx = hconflx * rescale_factor
```

Compute HXD rate

```
;       HXD/PIN
wv_hxdPIN=0.5*(hxd_PINwgrid[1:*]+hxd_PINwgrid)
;
lwvl=abs(ae2_lstr.WVL)
effar=(interpol(ea_hxdPIN,wv_hxdPIN,lwvl)>0) < max(ea_hxdPIN)
flxlin= linflx*effar            ;[ct/s]
;
cwvl=abs(ae2_hcstr.midWVL)
effar=(interpol(ea_hxdPIN,wv_hxdPIN,cwvl)>0) < max(ea_hxdPIN)
flxcon= hconflx*effar           ;[ct/s]
;
flx_hxdPIN = total(flxlin)+total(flxcon)

;       HXD/GSO
wv_hxdGSO=0.5*(hxd_GSOwgrid[1:*]+hxd_GSOwgrid)
;
lwvl=abs(ae2_lstr.WVL)
effar=(interpol(ea_hxdGSO,wv_hxdGSO,lwvl)>0) < max(ea_hxdGSO)
flxlin= linflx*effar            ;[ct/s]
;
cwvl=abs(ae2_hcstr.midWVL)
effar=(interpol(ea_hxdGSO,wv_hxdGSO,cwvl)>0) < max(ea_hxdGSO)
flxcon= hconflx*effar           ;[ct/s]
;
flx_hxdGSO = total(flxlin)+total(flxcon)

print,''
```

```
print,'HXD/PIN bremss count rate = ',flx_hxdPIN,'       total counts = ',flx_hxd
PIN*EXPTIME*1e3
print,'HXD/PIN background counts = ',(1.+cornorm)*total(bg_HXDPIN.COUNTS)*EXPTIM
E*1e3/bg_hxdPIN_exp
print,''
print,'HXD/GSO bremss count rate = ',flx_hxdGSO,'       total counts = ',flx_hxd
GSO*EXPTIME*1e3
print,'HXD/GSO background counts = ',(1.+cornorm)*total(bg_HXDGSO.COUNTS)*EXPTIM
E*1e3/bg_hxdGSO_exp
print,''
```

Compute XIS rate

```
;       XIS/FI
wv_XISFI=0.5*(XIS_FIwgrid[1:*]+XIS_FIwgrid)
;
lwvl=abs(ae2_lstr.WVL)
effar=(interpol(ea_XISFI,wv_XISFI,lwvl)>0) < max(ea_XISFI)
flxlin= linflx*effar            ;[ct/s]
;
cwvl=abs(ae2_hcstr.midWVL)
effar=(interpol(ea_XISFI,wv_XISFI,cwvl)>0) < max(ea_XISFI)
flxcon= hconflx*effar           ;[ct/s]
;
flx_XISFI = total(flxlin)+total(flxcon)

;       XIS/BI
wv_XISBI=0.5*(XIS_BIwgrid[1:*]+XIS_BIwgrid)
;
lwvl=abs(ae2_lstr.WVL)
effar=(interpol(ea_XISBI,wv_XISBI,lwvl)>0) < max(ea_XISBI)
flxlin= linflx*effar            ;[ct/s]
;
cwvl=abs(ae2_hcstr.midWVL)
effar=(interpol(ea_XISBI,wv_XISBI,cwvl)>0) < max(ea_XISBI)
flxcon= hconflx*effar           ;[ct/s]
;
flx_XISBI = total(flxlin)+total(flxcon)

print,''
print,'XIS/FI count rate = ',flx_XISFI,'        total counts = ',flx_XISFI*EXPTI
ME*1e3
print,'XIS/FI background counts = ',(1.+cornorm)*total(bg_XISFI.COUNTS)*EXPTIME*
1e3/bg_hxdPIN_exp
print,''
```

```
print,'XIS/BI count rate = ',flx_XISBI,'        total counts = ',flx_XISBI*EXPTI
ME*1e3
print,'XIS/BI background counts = ',(1.+cornorm)*total(bg_XISBI.COUNTS)*EXPTIME*
1e3/bg_hxdPIN_exp
print,''
```

Compute XIS spectra

To construct the Suzaku/XIS spectra, we use the line and continuum fluxes calculated above (LINFLX and CONFLX), rebin them into a consistent energy grid, and convolve with the response matrix.

```
;       XIS/FI

;       bin the spectra

egrid=12.3985/xis_FIwgrid & os=sort(egrid) & egrid=egrid[os] & tmp=egrid
lnrg=12.3985/abs(ae2_lstr.WVL) & os=sort(lnrg)
xisFI_lspc=     hastogram(lnrg[os],egrid,wts=linflx[os])        ;[ph/s/cm^2/bin]
;
egrid=12.3985/xis_FIwgrid & os=sort(egrid) & egrid=egrid[os]
cnrg=12.3985/ae2_cstr.WVL & os=sort(cnrg)
xisFI_cspc=     rebinw(conflx[os],cnrg[os],egrid,/perbin)       ;[ph/s/cm^2/bin]
;
xisFI_spc=      xisFI_lspc + xisFI_cspc

;       convolve with RMF
tmp=12.3985/xis_FIwgrid & ee_xisFI=0.5*(tmp[1:*]+tmp)
if strlowcase(xisFI_RMF) ne 'none' then $
  conv_rmf,egrid,xisFI_spc,xisFI_channel,xisFI_predct,rm_xisFI,$
  effar=ea_xisFI,nrgar=0.5*(ea_xisFIstr.ELO+ea_xisFIstr.EHI),$
  verbose=!VERBOSE

;       units are now in ct/s/bin, so multiply by EXPTIME
xisFI_predct = xisFI_predct * EXPTIME * 1e3     ;[ct/bin]

;       XIS/BI

;       bin the spectra
egrid=12.3985/xis_BIwgrid & os=sort(egrid) & egrid=egrid[os] & tmp=egrid
lnrg=12.3985/abs(ae2_lstr.WVL) & os=sort(lnrg)
xisBI_lspc=     hastogram(lnrg[os],egrid,wts=linflx[os])        ;[ph/s/cm^2/bin]
;
egrid=12.3985/xis_BIwgrid & os=sort(egrid) & egrid=egrid[os]
cnrg=12.3985/ae2_cstr.WVL & os=sort(cnrg)
xisBI_cspc=     rebinw(conflx[os],cnrg[os],egrid,/perbin)       ;[ph/s/cm^2/bin]
```

```
;
xisBI_spc=      xisBI_lspc + xisBI_cspc

;       convolve with RMF
tmp=12.3985/xis_BIwgrid & ee_xisBI=0.5*(tmp[1:*]+tmp)
if strlowcase(xisBI_RMF) ne 'none' then $
  conv_rmf,egrid,xisBI_spc,xisBI_channel,xisBI_predct,rm_xisBI,$
  effar=ea_xisBI,nrgar=0.5*(ea_xisBIstr.ELO+ea_xisBIstr.EHI),$
  verbose=!VERBOSE

;       units are now in ct/s/bin, so multiply by EXPTIME
xisBI_predct = xisBI_predct * EXPTIME * 1e3     ;[ct/bin]
```

Obtain Poisson deviates of source and background spectra

```
nFI=n_elements(xisFI_predct)
xisFI_counts=lonarr(nFI) & xisFI_countsbg=lonarr(nFI) & xisFI_countstot=lonarr(n
FI)
tmp=xisFI_predct & for i=0L,nFI-1L do if tmp[i] gt 1d-20 then $
        xisFI_counts[i]=randomu(seed,poisson=tmp[i])
;
tmp=bg_xisFI.counts*(EXPTIME*1e3)*(1.+cornorm)/BG_XISFI_EXP
for i=0L,nFI-1L do if tmp[i] gt 1d-20 then $
        xisFI_countsbg[i]=randomu(seed,poisson=tmp[i])
;
xisFI_countstot = xisFI_counts + xisFI_countsbg

nBI=n_elements(xisBI_predct)
xisBI_counts=lonarr(nBI) & xisBI_countsbg=lonarr(nBI) & xisBI_countstot=lonarr(n
BI)
tmp=xisBI_predct & for i=0L,nBI-1L do if tmp[i] gt 1d-20 then $
        xisBI_counts[i]=randomu(seed,poisson=tmp[i])
;
tmp=bg_xisBI.counts*(EXPTIME*1e3)*(1.+cornorm)/BG_XISBI_EXP
for i=0L,nBI-1L do if tmp[i] gt 1d-20 then $
        xisBI_countsbg[i]=randomu(seed,poisson=tmp[i])
;
xisBI_countstot = xisBI_counts + xisBI_countsbg
```

make plots

```
peasecolr & !p.multi=[0,1,2]

plot,xisFI_channel,xisFI_countstot,psym=10,thick=2,$
        xtitle='E [keV]',ytitle='counts',$
```

```
        title='XIS/FI '+strtrim(string(EXPTIME,'(f5.2)'),2)+' ks',$
        xrange=[0.2,8],/xs,/xlog,yrange=max(xisFI_countstot)*[1e-4,1],/ylog
oplot,xisFI_channel,xisFI_counts,psym=10,thick=2,col=1
oplot,xisFI_channel,xisFI_countsbg,psym=10,thick=2,col=2
oplot,xisFI_channel,xisFI_predct,col=3

plot,xisBI_channel,xisBI_countstot,psym=10,thick=2,$
        xtitle='E [keV]',ytitle='counts',$
        title='XIS/BI '+strtrim(string(EXPTIME,'(f5.2)'),2)+' ks',$
        xrange=[0.2,8],/xs,/xlog,yrange=max(xisBI_countstot)*[1e-4,1],/ylog
oplot,xisBI_channel,xisBI_counts,psym=10,thick=2,col=1
oplot,xisBI_channel,xisBI_countsbg,psym=10,thick=2,col=2
oplot,xisBI_channel,xisBI_predct,col=3
stample,stacol=4
!p.multi=0
```

```
Summary

-------------------------------------------------------------------------------
        Outputs are in the following variables:
-------------------------------------------------------------------------------

------------------------------------------------------------------------------
XIS/FI --
model spectrum, pre-RMF :       xisFI_spc [ph/s/cm^2/bin] (egrid,xis_FIwgrid)
total model count rate  :       flx_XISFI [ph/s]
predicted counts intensity :    xisFI_predct [ct/channel] (xisFI_channel)
simulated source counts :       xisFI_counts [ct/channel] (xisFI_channel)
simulated background counts :   xisFI_countsbg [ct/channel] (xisFI_channel)
simulated total counts  :       xisFI_countstot [ct/channel] (xisFI_channel)

------------------------------------------------------------------------------
XIS/BI --
model spectrum, pre-RMF :       xisBI_spc [ph/s/cm^2/bin] (egrid,xis_BIwgrid)
total model count rate :        flx_XISBI [ph/s]
predicted counts intensity :    xisBI_predct [ct/channel] (xisBI_channel)
simulated source counts :       xisBI_counts [ct/channel] (xisBI_channel)
simulated background counts :   xisBI_countsbg [ct/channel] (xisBI_channel)
simulated total counts  :       xisBI_countstot [ct/channel] (xisBI_channel)

-------------------------------------------------------------------------------
HXD/PIN --
bremss count rate :     flx_hxdPIN [ph/s]
```

```
------------------------------------------------------------------------------
HXD/GSO --
bremss count rate :     flx_hxdGSO [ph/s]
------------------------------------------------------------------------------

6. Comparison with PIMMS

;       Results using this thread  are roughly consistent with
;       PIMMS results. PIMMS returned the following XMM count rates
;       for a CHANDRA-ACIS-I count rate of 1 cts/s and a Raymond-Smith
;       model of temperature 0.485 keV:
;             SUZAKU XIS FI : 5.066E-01 cts/s
;             SUZAKU XIS BI : 1.085E+00 cts/s
;
;       Running this thread with  !ABUND = getabund('Allen'),
;       T_components = [6.8], and  obs_rate = 1.0 gives:
;             SUZAKU XIS FI : 0.482 cts/s
;             SUZAKU XIS BI : 0.986 cts/s
     ------------------------------------------------------------------


    pintofale()head.cfa.harvard.edu (LL/VK)
```

4.3 More PINTofALE Usage Tips and Reminders

4.3.1 Changing the Line Database using !LDBDIR

Using the `.par` files, changing the line database that is read in by running e.g., `rd_line.par` and `linespec.par` can easily be done by re-setting the `!LDBDIR` system variable. Note that when explicitly calling functions line `rd_line()`, a default is assumed (currently set to `$CHIANTI`). When changing databases, the thing to remember is that the data must be re-read in order for the new database emissivities to be used, eg

```
; Read line emissivities, save the line structure, then change DB and re-read

!LDBDIR='$CHIANTI'
@rd_line.par
linstr_chianti=rd_line_linstr

!LDBDIR='$SPEX'
@rd_line.par
linstr_spex=rd_line_linstr
```

The terms `$CHIANTI` and `$SPEX` are shorthand for the full paths to the CHIANTI and SPEX emissivity databases. These may be externally defined via the UNIX shell environment variables `CHIANTI_LDB` and `SPEX_LDB` respectively. If these variables are not set, then the default locations under `!TOPDIR/emissivity/...` are looked at. Further, if `!LDBDIR` is set to a pathname not beginning with "/" then it is assumed to point to a location relative to the current working directory. In this case, rerunning `initale.pro` will result in prepending `!TOPDIR` to `!LDBDIR`.

4.3.2 Changing Element Abundances and `!ABUND`

The default `!ABUND` variable contains the abundances of elements H-Zn as a fraction of the hydrogen abundance (by number), where the abundance of hydrogen is 1. The function `getabund()` is sufficiently flexible to return abundances in different units if desired (e.g., on the $\log_{10}$ scale with the hydrogen abundance set to 12), though other routines will not understand these units and so they should not be used for input into `PINTofALE` procedures and functions.

For the case of optically thin emission, element abundances are for practical purposes just scaling factors (though the abundance-dependent ratio N_e/N_H also enters into some calculations in `PINTofALE`). So, abundances can be changed and applied *without* re-reading the line emissivity database. The general way to perform calculations with different abundances is therefore to get the desired abundances into an array, like !ABUND, then run the calculations that take abundances as input. E.g.,

```
; Do line synthesis for solar photospheric, then metal poor [M/H]=-0.5
; abundances, assuming SP_WGRID contains wavelength grid bin boundaries.
; Default abundances on start-up are Grevesse et al. (1992) - re-read
; just in case

io_ww=sp_wgrid

!abund=getabund('Grevesse')
@linespec.par
linespec_0.0=varsmooth(linespec_spec,5.,type='beta=1.8')

!abund=getabund('Grevesse',metals=-.5)
@linespec.par
linespec_ne13=varsmooth(linespec_spec,5.,type='beta=1.8')
```

This is actually not the best example of changing abundances, since the `linespec()` routine can re-read in the line emissivities itself. However, if run as above, `linespec()` will re-use the emissivities already read in from the first call of `linespec.par`.

4.3.3 Changing the Plasma Density and `!EDENS`

The default plasma density value is stored in the `!EDENS` system variable. All `.par` files in the `.../ardb/` directory that need the plasma density as an input use `!EDENS`. The plasma line emissivities in the `PINTofALE` databases are stored on a default 81 temperature grid (`!LOGT`) for single density values in the range 10^8-$10^1$4 cm^{-3}. Thus, functions such as `rd_line()` only read in emissivities at a single density value, and this value is the nearest database value to the input requested density. Thus setting `!EDENS=1.2e10` will result in emissivities for `1.0e10`. In the example sessions in sections 4.1.1 and 4.1.2, we show how to easily linearly interpolate to a different density value (though this should rarely be required).

The important thing to remember about re-setting the plasma density is that the database needs to be re-read and operations performed on the data read in for the particular density. In this way, unlike `!ABUND`, `!EDENS` is at a similar `PINTofALE` level as `!LDBDIR`. For example:

```
; do spectrum synthesis for two different densities, assuming SP_WGRID
; contains wavelength grid bin boundaries.

io_ww=sp_wgrid

delvar,linespec_linstr ; makes linespec re-read emissivity DB
!EDENS=1.e10
@linespec.par
linespec_ne10=varsmooth(linespec_spec,5.,type='beta=1.8')

delvar,linespec_linstr
!EDENS=1.e13
@linespec.par
linespec_ne13=varsmooth(linespec_spec,5.,type='beta=1.8')
```

4.3.4 More help

Almost all `PINTofALE` routines are self-documenting. That is, when called with no arguments, they print the usage syntax and quit. A list of all routines in the current release is listed in the file `doc/synopsis.txt` . Detailed help on the inputs and outputs for each routine are in the header of each file, and are collected together in a single HTML file at `doc/PoA.html` .

Some useful online documents are listed below:

- `http://hea-www.harvard.edu/PINTofALE/doc/synopsis.txt`
  – up-to-date list of the routines in the current release
- `http://hea-www.harvard.edu/PINTofALE/doc/PoA.html`
  – up-to-date description of `PINTofALE` routines, inputs and outputs
- `http://hea-www.harvard.edu/PINTofALE/#help`
  – list of online guides and analysis threads
- `http://hea-www.harvard.edu/PINTofALE/doc/LINEID.howto`
  – guide to using the GUI-based line identification program, `LINEID()`
- `http://hea-www.harvard.edu/PINTofALE/doc/FITLINES.howto`
  – guide to using the GUI-based line fitting program, `FITLINES()`
- `http://hea-www.harvard.edu/PINTofALE/doc/MCMC_DEM.html`
  – guide to using the Markov-Chain Monte Carlo algorithm to reconstruct stellar coronal differential emission measures

4.3.5 IDL tips

RETALL and disappearing variables

As usual, when a routine fails or crashes (due to a missing or undefined but necessary variable, or due to a bug), or is deliberately halted (with a ctrl-C, or with a `STOP` or `BREAKPOINT`) the user should be careful concerning the level at which the IDL prompt is returned—whether one is still within a procedure or function, or back to the IDL `$MAIN$` level. Variables defined only within a procedure or function will not be accessible outside the routine unless specifically passed. Many options are available at this stage: `RETALL` will return the user to the `$MAIN$` program level; `.OUT` will pop to the level immediately above; and `RETURN` will also pop out from within a procedure.

We have deliberately eschewed the use of IDL's native error-catching via `on_error`, etc., because we find that it takes away from one of IDL's biggest strengths – the ability to pinpoint quickly and painlessly where something has gone wrong, and the capacity to recover from bad inputs *in situ*. This of course usually requires a somewhat detailed knowledge of the innards of the offending program, but on the other hand provides an easy way for users to make local enhancements or changes as needed.

Color Plots

We have developed this package entirely on 8-bit color depth monitors, and consequently many of the color settings are optimized for such setups. Indeed, we are aware that many of these setups do not work as expected on 24-bit color screens. For now, we recommend setting the X resources

```
idl.visual:     PseudoColor
idl.colors:     250
```

prior to starting an IDL session. For 24-bit screens which have more than 256 colors, we recommend setting the keyword DECOMPOSED=0 in calls to `device` procedure. This ensures that the color tables are loaded as {R,G,B} triplets (see David Fanning's guide on using color tables on a 24-bit display, `http://www.dfanning.com/color_tips/noxloadct.html`).
This keyword is automatically in most cases, but users are cautioned to be aware that they will encounter cases where it may not be.

A useful routine in the `PINTofALE` suite is `setkolor`, which sets named colors to specific indices in an existing color table. Usage is demonstrated, for instance, in the example in §3.1.2 and the detailed walk-through session in §4.1.1. (Note that this was originally called `setcolor`, but the name has now been changed to avoid conflict with a pre-existing JHUAPL routine of the same name.)

Note that at initialization, color table 3 is first loaded in, and a series of specific colors are loaded into color indices 1-79 using the routine `peasecolr`.

```
Color Indices

(for black background)
BLUE:   1,10-19
RED:    2,20-29
GREEN:  3,30-39
BROWN:  4,40-49
PINK:   5,50-59
CYAN:   6,60-69
YELLOW: 7,70-79
SAFFRON: 8,80-89

(for white background)
BLUE:   1,10-19
RED:    2,20-29
```

```
GREEN:  3,30-39
YELLOW: 4,40-49
PINK:   5,50-59
CYAN:   6,60-69
BROWN:  7,70-79
SAFFRON: 8,80-89
```

These colors can be overridden at any time using LOADCT, etc., though we recommend that that not be done because color indices in most routines are optimised for this color table.

Part I

Appendices

Appendix A

References

A.1 REFERENCES

Argiroffi, C., Maggio, A., & Peres, G., 2003, A&A, 404, 1033

Arnaud, K.A., 1996, Astronomical Data Analysis Software and Systems V, Eds.G.Jacoby and J.Barnes, p17, ASP Conf.Ser., v.101

Ball, W., Drake, J.J., Lin, L., Kashyap, V., 2004, in preparation

Balucinska-Church, M., McCammon, D., 1992, ApJ 400, 699

Cash, W., 1979, ApJ, 228, 939

Cruddace, R., Paresce, F., Bowyer, S., Lampton, M., 1977, ApJ 187, 497

Dere, K.P., Landi, E., Mason, H.E., Monsignori-Fossi, B.C., & Young, P.R., 1997, A&AS, 125, 145

Drake, et al., 2004, in preparation

Drake, J.J., Kashyap, V., ApJ, 2001, 547, 428

Freeman, P.E., Doe, S., & Siemiginowska, A., 2000, in *Astronomical Data Analysis Software and Systems X*, Eds. F.A.Primini, F.R.Harnden,Jr., and H.E.Payne, in press.

Garcia, D., Drake, J.J., Lin, L., Kashyap, V., 2004, in preparation

Grevesse, N., Sauval, A.J., 1999, A&A, 347, 348

Judge, P.G., Hubeny, V., & Brown, J.C., 1997, ApJ, 475, 275

Kaastra, J.S., Mewe, R., & Nieuwenhuijzen, H., 1996, in UV and X-ray Spectroscopy of Astrophysical and Laboratory Plasmas, Eds.K.Yamashita and T.Watanabe, p.411, Univ.Acad.Press

Kashyap, V., & Drake, J.J., 2000, BASI, 28, 475

Kashyap, V., & Drake, J.J., 1998, ApJ, 503, 450

Kashyap, V., Rosner, R., Micela, G., Sciortino, S., Vaiana, G. S., Harnden, F.R., Jr., 1992, ApJ 391, 684

Maggio, A., Drake, J.J, Kashyap, V., Micela, G., Sciortino, S., Peres, G., Harnden, F.R.,Jr., Murray, S., 2003, in Stellar Coronae in the Chandra and XMM-NEWTON Era, Eds. F.Favata and J.Drake, pg.57, ASP Conf.Ser., v.277

McLaughlin, B.M., Kirby, K.P., 1998, J. Phys. B: At. Mol. Opt. Phys., 31, 4991

Megropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E., 1954, J.Chem.Phys., 21, 1087

Mewe, R., Lemen, J.R., van den Oord, G.H.J., 1986, A&ASS, 65, 511

Morrison, R., & McCammon, D., 1983, ApJ 270, 119

Neal, R.M., 1993, Technical Report CRG-TR-93-1, Dept.Comp.Sci., Univ.Toronto

Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., 1998 ,*Numerical Recipes in C*, New York, NY: Cambridge University Press

Rumph, T., Bowyer, S., & Vennes, S., 1994, AJ 107, 2108

Verner, D.A., & Yakovlev, D.G., 1995, A&AS, 109, 125

Wilms, J., Allen, A., McCray, R., 2000, ApJ, 542, 914

Appendix B

List of `PINTofALE` Routines and Scripts

This appendix lists the *released* `PINTofALE` routines. Other routines that have been incorporated since a major release, that have been contributed by other users but have not yet been incorporated, or that are under development but have been made available, may be available—check the web site if you are interested in routines that do things that are not listed here, or send us e-mail and we will see if we can develop them ourselves.

B.1 POA_CONSTRUCTOR [/scrypt]

```
script  PoA_constructor
        this is a "front-end" to the initialization script INITALE of
        PINTofALE, needed for older versions of IDL, for which system
        variables need to be defined before running ANY procedure which
        uses them.

        this script defines the system variables with default values
        and then calls INITALE.PRO to complete the initialization.

syntax
        @PoA_constructor

variables
        for a complete list of the PINTofALE system variables, see
        the description for INITALE.PRO .

calls subroutines
        INICON
        INITALE

restrictions
        it is assumed that this is called only once per session, and
        that it is called _first_

history
        Vinay Kashyap (FebMMI; based on suggestion by A.Maggio)
        added !ATOMDB and !APECDIR (VK; Jun02)
```

B.2 ADDRSP [/util]

```
function        addrsp
        combine two response matrices to make a composite RSP and
        return a structure in the same format as RD_OGIP_RMF()

        steps through each row of an RMF and tacks on the array
        from a 2nd RMF, updating the N_GRP, N_CHAN, F_CHAN elements
        as appropriate.

syntax
        rsp=addrsp(rm1,rm2,ea1=ea1,ea2=ea2,eps=eps,verbose=verbose)

parameters
        rm1     [INPUT; required] response matrix structure as
                read in by RD_OGIP_RMF
        rm2     [INPUT; required] response matrix structure to
                be added to RM1
                * RM1 and RM2 are expected to have the following fields:
                NNRG,ELO,EHI,NCHAN,EMN,EMX,N_GRP,F_CHAN,N_CHAN,MATRIX,FIRSTCHAN

keywords
        ea1     [INPUT] if set to a scalar or size matches RM1.ELO,
                then multiplies RM1 by EA1 prior to addition
        ea2     [INPUT] as EA1, but for RM2
                * default for EA1 and EA2 is 1.0
        eps     [INPUT] a small number, below which to set everything
                to zero
                * default is 1e-6
        morecol [INPUT] designed as a buffer so that a matrix won't
                overflow the bounds automatically set.  by default,
                the new matrix is defined at an early stage to be
                max(RM1.N_CHAN)+max(RM2.N_CHAN)+100+MORECOL.
                * note that MORECOL can even be negative.
                * use wisely.
        verbose [INPUT] controls chatter
        ea3     [JUNK] here just to prevent user error
        ea4     [JUNK] here just to prevent user error
        ea5     [JUNK] here just to prevent user error
        ea6     [JUNK] here just to prevent user error
                * if EA3, EA4, EA5, or EA6 are set, the program will
                  complain and quit.  this is done to avoid the obvious
                  user error of trying to specify EA3=EA3, etc. if
                  adding a third RMF/ARF to a preexisting one.
```

```
        _extra  [JUNK] here only to avoid crashing the program

subroutines
        KILROY
        OGIPZIP

history
        vinay kashyap (Nov2001)
        bug correction re max possible matrix elements when NGROUP>>1;
          added dummy keywords EA3, EA4, EA5, EA6 and extra checks
          against tricky inputs (VK; Aug02)
        bug correction: wasn't handling N_GRP=1 (VK; Aug04)
```

B.3 ADJUSTIE [/fitting]

```
procedure       adjustie
        adjust the parameters to account for any ties among them, and
        return the corrected set

syntax
        adjustie,a,ties=ties,vname=vname

parameters
        a       [I/O; required] parameters to be updated

keywords
        ties    [INPUT] string array describing how a parameter is tied to
                other parameters.
                * must be a mathematical expression of the form
                  "a0 = a1/10.", "a2=a3+a4", "a5 = a5 < 0", etc.
        vname   [INPUT; default='a'] in case TIES contains variable names
                such as "p0=p2^2", etc, then set VNAME='p'

                NOTE:  the following keywords are generated in situ the first
                time the function is called.  on subsequent calls they are
                used instead of EXECUTEing TIES in order to improve speed.

                NOTE2: actually, they are not implemented yet.  someday.

        ifit    [I/O] integer array describing which parameters are frozen
                (0's) and which are not (1's).  updated upon exit to include
                information given in TIES
        itie    [I/O] integer array showing which parameters are tied.
        tieto   [I/O] linked list showing which parameters each of the
                ITIE are tied to which parameter
        tieops  [I/O] as TIETO, but containing information on which operator
                is used, e.g., "+", "-", "*", "/", "<", ">", etc.
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (Oct98)
        converted to IDL5 (VK; OctMM)
```

B.4 ALALOESS [/util]

```
function        alaloess
        computes a loess/lowess type smoothed function for the
        given set of input points by fitting polynomials over
        the neighborhood of each point and using the local value
        of the polynomial fit as the smoothed curve

        another one in a hoary line of PINTofALE smoothing routines,
        joining SMOOTHIE, NOISMOOTH, VARSMOOTH, VOORSMOOTH, REGROUP,
        SPLAC, CONV_RMF, SCRMF, UNKINK, LINEREM, HAARLINE, etc.

syntax
        yl=alaloess(yy,xx,hwidth=hwidth,ysig=ysig,ndeg=ndeg,$
        yerr=yerr,verbose=vv)

parameters
        yy      [INPUT; required] the points that need to be smoothed
        xx      [INPUT] the abscissae for which YY are defined
                * if size eq size of YY or +1, taken to be the midpoints
                  or the grid
                * if insensible, taken to be array indices, LINDGEN(N(YY))

keywords
        hwidth  [INPUT] half range over which local polynomial fits are
                to be performed
                * if XX are array indices, HWIDTH is forced to be >0
                * if XX is a predefined grid, the default value is the
                  minimum of delta(XX)
        ysig    [INPUT] error on YY
                * if size matches that of YY, then maps 1-to-1
                * if scalar, taken to be
                    <0 : absolute error
                    >0 and <1 : fractional error
                    >1 : percentage error
                    =0 : ignored in call to POLY_FIT()
                  on YY
                * otherwise, assumed to be sqrt(abs(YY)+0.75)+1
        ndeg    [INPUT] degree of the polynomial to be fit
                * default is 3
        yerr    [OUTPUT] 1-sigma error at the smooth location from
                each fit.
                * WARNING: this is to be taken only as an illustrative
                  number and not used for anything quantitative, because
                  the errors in adjacent bins are not independent
```

```
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

example
        .run alaloess   ;(requires PEASECOLR)

these are all the smoothing tools in PINTofALE
        ALALOESS() makes a loess curve
        CLTSMOOTH() removes -ves from background subtracted spectra or light curves
        CONV_RMF convolves with specified RMF
        HAARTRAN() smoothes by filtering on Haar wavelet coefficients
        LINEREM() removes lines from spectrum by iterative S/N filtering
        NOISMOOTH() does boxcar accumulation a la Ebeling's asmooth
        REGROUP() accumulates from one end
        SCRMF does fast convolution using optimized RMF
        SMOOTHIE does peak-descent accumulation to build up S/N
        SPLAC computes a segmented piecewise linear approximations
        UNKINK() removes sharp kinks from a curve
        VARSMOOTH() does local smoothing at specified scales
        VOORSMOOTH() does fast local smoothing at specified scales

history
        vinay kashyap (MarMMX)
        some bug fixes, and hardening to poly_fit failures -- where it fails
          now simply keep old value (VK; MayMMX)
        updated to be more robust to NaNs (VK; AprMMV)
```

B.5 APEDANCE [/]

```
procedure       apedance
        remove or correct the abundance dependance of APED emissivities

        APED emissivities as stored on disk include both ion balance
        (Mazzotta et al.) and abundances (Anders & Grevesse).  This
        can be inconvenient in some PINTofALE tasks since other line
        emissivity databases in PINTofALE do not include either factor.
        It is not possible to remove the effects of the included ion
        balance, but removing the abundance dependance is simply a
        matter of dividing the emissivity of each line by the abundance
        appropriate to the element producing that line.

syntax
        apedance,line,Z,abund=abund,apabref=apabref,verbose=verbose

parameters
        line    [I/O] APED line emissivities
                * emissivity structure out of RD_LINE()
                * if 2D array, assumed to be an array of size (T,Wvl)
        Z       [INPUT] atomic numbers
                * required if LINE is an array and not a structure
                * must match size of 2nd dimension in LINE

keywords
        abund   [INPUT] if given, updates LINE by multiplying by
                the ratio of ABUND/(Anders & Grevesse)
                * the default is to just divide by Anders & Grevesse
                  abundances, effectively removing the abundance
                  dependance from the emissivities
        apabref [INPUT] if set to a reference other than Anders & Grevesse,
                it is assumed that the input LINE emissivities need to be
                corrected according to the specified abundances.
                * WARNING: do not set this unless you know exactly
                 what you are doing
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to avoid crashing the program

warning
        no checks are made to verify that the emissivities have
        have already had the abundances taken out, or that the
        abundances are corrected according to some other baseline.
        USE CAREFULLY!
```

```
history
        vinay kashyap (Apr2004)
```

B.6 ARITHTOGRAM [/util]

```
function        arithtogram
        return the result of an arithmetical operation on the
        frequency distributions of two lists

syntax
        hx=arithtogram(x1,x2,xout,operator,/plus,/minus,/divide,$
        w1=w1,w2=w2,nbin=nbin,xmin=xmin,xmax=xmax,/nonorm)

parameters
        x1      [INPUT; required] list of numbers whose frequency
                distribution must be multiplied with that of X2
        x2      [INPUT; required] list of numbers whose frequency
                distribution must be multiplied with that of X1
                * it is assumed that X1 and X2 both have the same units
                  because otherwise this kind of multiplication makes
                  no sense
        xout    [OUTPUT; required] the list of numbers at which the
                output is tabulated
                * by default, this is the sorted unique set of [X1,X2]
                * if NBIN is set, generates a logarithmic or linear
                  grid of that size
        oper    [INPUT] must be one of '+', '-', '/', '*':
                the arithmetical operation to carry out on the
                frequency distributions of X1 and X2, i.e.,
                        result = X1 OPER X2
                * if not given, assumed to be '*', unless one of
                  keywords PLUS, MINUS, or DIVIDE are set

keywords
        plus    [INPUT] if set, OPER is assumed to be '+'
        minus   [INPUT] if set, OPER is assumed to be '-'
        divide  [INPUT] if set, OPER is assumed to be '/'
                * DIVIDE takes precedence over MINUS takes precedence
                  over PLUS
        w1      [INPUT] optional weight to be applied to histogram(X1)
        w2      [INPUT] optional weight to be applied to histogram(X2)
                * if not given, W1 and W2 are assumed to be 1
        nbin    [INPUT] number of bins in the output
                * if not set, will be the sorted unique set of [X1,X2]
                * if -ve, will produce logarithmic gridding
        xmin    [INPUT] minimum value in the output grid
                * by default, uses min(X1)>min(X2)
        xmax    [INPUT] maximum value in the output grid
```

```
                * by default, uses max(X1)<max(X2)
        nonorm  [INPUT] if set, does not make a correction for the
                width of the bins in XOUT.
        _extra  [JUNK] here only to prevent crashing the program

description
        this program does not create histograms as intermediate products
        but rather produces an operation at the best available data
        resolution.  this is of great use in Monte Carlo calculations
        or in creating probability distributions in MCMC.  the algorithm
        is straightforward: build up a cdf for each list, and interpolate
        onto a common grid, and multiply, divide, add, or subtract the
        d(cdf)'s, suitably normalized for the bin widths and number of
        elements.

example
        x1=randomn(seed,10000L) & x2=randomn(seed,10000L)+2.
        xmin=-2. & xmax=3. & nbin=101L
        hh=arithtogram(x1,x2,xout,'*',xmin=xmin,xmax=xmax,nbin=nbin,/nonorm)
        hp=arithtogram(x1,x2,xout,'+',xmin=xmin,xmax=xmax,nbin=nbin,/nonorm)
        hd=arithtogram(x1,x2,xout,'/',xmin=xmin,xmax=xmax,nbin=nbin,/nonorm)
        hm=arithtogram(x1,x2,xout,'-',xmin=xmin,xmax=xmax,nbin=nbin,/nonorm)
        h1=histogram(x1,min=xmin,max=xmax,binsize=median(xout[1:*]-xout))
        h2=histogram(x2,min=xmin,max=xmax,binsize=median(xout[1:*]-xout))
        plot,xout,hh,psym=10 & oplot,xout,h1,col=2 & oplot,xout,h2,col=3
                oplot,xout,h1*h2,psym=10,col=4
        plot,xout,hp,psym=10 & oplot,xout,h1,col=2 & oplot,xout,h2,col=3
                oplot,xout,h1+h2,psym=10,col=4
        plot,xout,hd,psym=10 & oplot,xout,h1,col=2 & oplot,xout,h2,col=3
                oplot,xout,float(h1)/float(h2),psym=10,col=4
        plot,xout,hm,psym=10 & oplot,xout,h1,col=2 & oplot,xout,h2,col=3
                oplot,xout,h1-h2,psym=10,col=4

history
        vinay kashyap (Jun2005)
```

B.7 ARRAYEQ [/util]

```
function        arrayeq
        checks whether two input arrays are equal, and returns
        1: if they are identical
        0: if at least one element differs past tolerance
        -1: if sizes differ, but one array is a simple subset of the other

syntax
        AeqB=arrayeq(A,B,tol=tol)

parameters
        A       [INPUT; required] first array
        B       [INPUT; required] second array
                * arrays must be of some >numerical< type

keywords
        tol     [INPUT; default: 1e-4] tolerance within which two numbers
                are said to be identical
        _extra  [JUNK] here only to avoid crashing the program

history
        vinay kashyap (Nov98)
        converted to IDL5 notation (VK; OctMM)
```

B.8 B11001001 [/util]

```
function        b11001001
        returns the 0s and 1s of a byte variable

syntax
        d=b11001001(b,/str,/otto)

parameters
        b       [INPUT; required] scalar or array of bytes

keywords
        str     [INPUT] if set, returns the 0s and 1s in a string
                * overrides OTTO
        otto    [INPUT] if set, returns an 8xN(B) byte array as output

history
        vinay kashyap (Jul97)
```

B.9 BAMABS [/]

```
function        bamabs
        return photoelectric absorption cross-sections in energy range
        30 eV - 10 keV using the polynomial fit coefficients determined
        by Monika Balucinska-Church and Dan McCammon (Balucinska-Church,
        M.\ and McCammon, D.\ 1992, ApJ 400, 699).  This is an update
        to Morrison & McCammon 1983.

syntax
        sigabs=bamabs(w,abund=abund,/ikeV,/Fano,verbose=v)

parameters
        w       [INPUT; required] photon energy values at which to compute
                the absorption
                * default units are [Ang], unless IKEV is set, when they are
                  assumed to be [keV]

keywords
        abund   [INPUT] abundances relative to H=1
                * default is to use Anders & Grevesse 1982
        ikeV    [INPUT] if set, assumes that W are in units of keV
        Fano    [INPUT] if set, the 4 strongest auto-ionizing resonances
                of He I are included; numbers come from Oza (1986, Phys
                Rev A, 33, 824), and Fernley et al. (1987, J.Phys B 20, 6457).
        noHeH   [INPUT] if set to any number other than 1 or 2, excludes
                H and He from the cross-sections
                * if set to 1, only excludes H
                * if set to 2, only excludes He
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to avoid crashing the program

subroutines
        GETABUND
        INICON

restrictions
        works only in energy range 30 eV - 10 keV
        be warned that the edges are idealized and thus unrealistic
          for high resolution spectra because they do not include
          any resonance structure

history
        vinay kashyap (OctMM; based on TOTLXS.FOR and XSCTNS.FOR of
          Balucinska-Church & McCammon, obtained from ADC catalog VI/62A,
```

```
  ftp://adc.gsfc.nasa.gov/pub/adc/archives/catalogs/6/6062A/ )
bug corrections (Brian Kern, 5Dec2001): Ne cross-section was
  missing "^3"; Fano calc had EPSj[j] (VK; Dec2001)
force calculations only in valid energy range; added keyword
  NOHEH (VK; Jun03)
bug correction: "exclude=0" means "include" (VK; Jul03)
bug correction: lambda selection for He (Andy Ptak; VK Jun05)
edge lambda correction: O, N, C changed acc. to numbers in
  Atomic Data Booklet (2nd Edition), Thompson, A., et al., 2001,
  LBL/PUB-490 Rev. 2 (Jeremy Drake; VK Sep2006)
```

B.10 BEHR_HUG [/stat]

```
function        BEHR_hug
        IDL wrapper to Bayesian Estimate of Hardness Ratios (BEHR)
        returns a structure containing the relevant outputs neatly
        summarized into fields for IDL consumption

        references:

        "Bayesian Estimation of Hardness Ratios: Modeling and Computations",
          Park, T., Kashyap, V.L., Siemiginowska, A., van Dyk, D., Zezas, A.,
          Heinke, C., and Wargelin, B., 2006, ApJ, 652, 610

        "BEHR: Bayesian Estimation of Hardness Ratios", Park, T., Kashyap, V.,
          Zezas, A., Siemiginowska, A., van Dyk, D., Connors, A., and Heinke, C.,
          2005, at Six Years of Science with Chandra Symposium, Nov 2-5, 2005, #5.6

        "Computing Hardness Ratios with Poissonian Errors",
          van Dyk, D.A., Park, T., Kashyap, V.L., \& Zezas, A.,
          2004, AAS-HEAD #8, 16.27
          http://www.aas.org/publications/baas/v36n3/head2004/137.htm

syntax
        behr=behr_hug(softsrc,hardsrc,softbkg,hardbkg,softarea,hardarea,$
        softeff=softeff,hardeff=hardeff,softidx=softidx,hardidx=hardidx,$
        softscl=softscl,hardscl=hardscl,softtbl=softtbl,hardtbl=hardtbl,/post,$
        level=level,algo=algo,details=details,nsim=nsim,nburnin=nburnin,$
        /hpd,nbins=nbins,outputf=outputf,outputR=outputR,outputHR=outputHR,$
        outputC=outputC,outputMC=outputMC,outputPr=outputPr,BEHRdir=BEHRdir,$
        verbose=verbose)

parameters
        softsrc [INPUT; required] counts in source region in soft (S) band
        hardsrc [INPUT; required] counts in source region in hard (H) band
        softbkg [INPUT; required] counts in background region in S band
        hardbkg [INPUT; required] counts in background region in H band
        softarea[INPUT; required] background scaling factor in S band
        hardarea[INPUT; required] background scaling factor in H band
                * (background region area)/(source region area)
                * can also include differences in exposure time into
                  the ratio, in the same manner as geometric area

keywords
        softeff [INPUT] effective area in soft (S) band
                * if not set, then set equal to HARDEFF if given, or
```

```
          1 otherwise
hardeff [INPUT] effective area in hard (H) band
        * if not set, then set equal to SOFTEFF if given, or
          1 otherwise
        * can also be the effective area relative to some
          special point on the detector (e.g., aimpoint)
          or even some specific detector (e.g., ACIS-I v/s ACIS-S)
softidx [INPUT] index of prior on S (range = 0+)
        * if not set, then set equal to HARDIDX if given, or
          0.5 otherwise
hardidx [INPUT] index of prior on H (range = 0+)
        * if not set, then set equal to SOFTIDX if given, or
          0.5 otherwise
        * similar to AGAMMA of PPD_SRC()
softscl [INPUT] scaling index of prior on S
        * if not set, then set to HARDSCL if given, or
          0 otherwise
hardscl [INPUT] scaling index of prior on H
        * if not set, then set to SOFTSCL if given, or
          0 otherwise
        * similar to BGAMMA of PPD_SRC()
softtbl [INPUT] filename containing a tabulated prior for S
hardtbl [INPUT] filename containing a tabulated prior for H
        * the table prior must be an ascii file with the following format:
          line 1: number of entries, say NLIN
          line 2: labels for the columns, ignored
          lines 3..NLIN+2: two whitespace separated columns of numbers,
            with each row containing the source intensity and the posterior
            density, in that order
        * the default filenames are "./tblprior_{soft|hard}.txt"
        * the default filenames are used iff SOFTTBL and HARDTBL are set
          but do not appear to exist
        * WARNING: if regex is used in the filename specification, only the
          first file from the list will be used.  furthermore, if specified,
          the table priors are applied to _all_ SOFTSRC and HARDSRC
post    [INPUT] if set, suggests the values of (SOFTIDX,SOFTSCL)
        and (HARDIDX,HARDSCL) going forward, i.e., what you should
        set the priors to in your next calculation for the same
        source
level   [INPUT] percentage confidence level at which to report error
        (default = 68)
details [INPUT] compute various ratios (true/false)?
        (default = true)
algo    [INPUT] calculation method, GIBBS (default) or QUAD
        * if set to AUTO or AUTO=N, then uses GIBBS for any
```

```
                  case where {SOFT|HARD}SRC > N, and QUAD below that
                  unless one of {SOFT|HARD}SRC > 99, in which case
                  GIBBS is set automatically
                  - if N is not set, assumed to be 15
                  - N can be set to any integer less than 100
        nsim    [INPUT] number of draws if algo=gibbs (default=10000)
        nburnin [INPUT] number of burn-in draws if algo=gibbs
                (default=5000 or NSIM/2, whichever is smaller)
        nbins   [INPUT] number of bins in integration if algo=quad (default=500)
        hpd     [INPUT] if set, computes the highest posterior density interval,
                which also is the smallest interval that includes the mode.
                * this is set by default
                * only has an effect if ALGO=QUAD
        outputF [INPUT] root of filename in which to place output
                (default = 'none')
                * output will be placed in the file, OUTPUTF.txt
                * unless OUTPUTF='none', in which case nothing is written out,
                    unless one of OUTPUT{R|C|HR|MC|PR} are set, in which case
                    the corresponding output is put in file BEHR_{R|C|HR|MC|PR}.txt
        outputR [INPUT] if set, writes output for R in OUTPUTF_R.txt
        outputC [INPUT] if set, writes output for C in OUTPUTF_C.txt
        outputHR[INPUT] if set, writes output for HR in OUTPUTF_HR.txt
        outputMC[INPUT] if set, writes Monte Carlo draws for lamS and lamH
                to OUTPUTF_draws.txt when algo=gibbs
        outputPr[INPUT] if set, writes the probability distributions for
                R, HR, C, lamS, and lamH to OUTPUTF_prob.txt when algo=quad
        BEHRdir [INPUT] full path to directory where BEHR executable resides
                (default = '/fubar/kashyap/AstroStat/BEHR')
        verbose [INPUT] controls chatter

requirements
        BEHR executable must be installed in BEHRDIR
        BEHR should be executable under the shell via SPAWN
        BEHR output assumed to be compatible with 12-19-2005 version

side-effects
        potentially creates numerous ascii files in $cwd

history
        vinay kashyap (SepMMV)
        bugfix: output file format was overflowing; force NBURNIN < NSIM;
          outputF for 1st iteration; added more references (VK; NovMMV)
        added keywords SOFTSCL,HARDSCL,POST,HPD; changed DETAILS default
          to true, OUTPUTF default to none; added extra fields to output
          structure; added ALGO option AUTO (VK; DecMMV)
```

```
altered behavior of ALGO=AUTO to use GIBBS if counts in one of the
  bands exceeds 100 (VK; OctMMVI)
added keywords SOFTTBL and HARDTBL (VK; FebMMVIII)
updated default BEHRdir (VK; OctMMXV)
```

B.11 BINERSP [/util]

```
function        binErsp
        returns position indices of input energ(y/ies) appropriately
        binned into the specified binning scheme

        returns -1 where E is outside the bin range

syntax
        iE=binersp(nrg,binmn,maxbin,bstr=bstr,nbin=nbin)

parameters
        nrg     [INPUT; required] photon energies for which to find
                the binning numbers
        binmn   [INPUT] array of bin minimum values
                * overrides all keywords
                * if not given, see description below on how it is
                  derived from BSTR.ELO, and NBIN & NRG.
        binmx   [INPUT] maximum of the bin max values.
                * if not a scalar, then the last element of array is used
                * see description below for how it is determined if
                  not given.

keywords
        bstr    [INPUT] structure containing the following pieces of
                information, as obtained from an OGIP-compliant response
                matrix (see RDRESP.PRO/RDARF.PRO): {NNRG, ELO, EHI}
        nbin    [INPUT; default=101] number of equally spaced bins into
                which to split the input array of photon energies
                * if -ve, binning will be logarithmic!
        _extra  [JUNK] here only to prevent crashing the program

description
        if BINMN is not given, then BINMN=BSTR.ELO
          if BSTR is also not given (or is incorrect), then
                BINMN=FINDGEN(ABS(NBIN))*DE+EMIN
          where DE=(EMAX-EMIN)/(ABS(NBIN)-1), EMIN=MIN(EE,MAX=EMAX),
          and EE=NRG if NBIN>0, EE=ALOG10(NRG) if NBIN<0
        if BINMX < MAX(BINMN), BINMN is truncated
        if BINMX < MIN(BINMN), BINMX is reset to MIN(BINMN)
        if BINMX is not given, then BINMX=MAX(BINMN)+BINMX(LAST)-BINMX(LAST-1)
          if BINMN is not given, BINMX=MAX(BSTR.EHI)
            if BSTR is also not given (or is incorrect), then BINMX=EMAX

history
```

```
vinay kashyap (Jul97)
converted to IDL5 notation (VK; OctMM)
```

B.12 BLAND [/]

```
function        bland
        returns a "blending factor" for each line in the emissivity list,
        defined as the ratio of the flux in the line to that within the
        given range.  (i.e., small values mean heavily blended lines)

syntax
        b=bland(flstr,dwvl,flx=flx,wflx=wflx,arstr=arstr,$
        dem=dem,abund=abund,effar=effar,wvlar=wvlar,$
        /temp,/noph,/ikev,/regrid)

parameters
        flstr   [INPUT; required] structure containing emissivity and
                wavelength information.  see RD_LINE for details.
                * ion-balance assumed to be included
                * if FLX and WFLX are set on input, FLSTR is ignored --
                  still required, but ignored.
        dwvl    [INPUT; default=median(delta(FLSTR.WVL))] the spectral
                "resolution" -- any line nearer than this value at any
                given wavelength is considered to be blended in to this.
                * if array of size FLSTR.WVL (or twice that size), set to
                  different values (with possible asymmetric ranges) at
                  each wavelength.

keywords
        arstr   [INPUT] structure of structures containing effective
                area information (see ARIA or GRATFLX)
                * if given in the right format, overrides call to LINEFLX
                  with call to GRATFLX
        flx     [I/O] contains the calculated flux due to each line
                * may be given on input and if WFLX is also set, FLSTR and
                  ARSTR are ignored
        wflx    [INPUT] if size matches that of FLX, assumed to be the
                wavelengths of the lines.
                * if FLX and WFLX are set on input, FLSTR and ARSTR are
                  ignored.
        _extra  [INPUT] pass defined keywords to
                GRATFLX (DEM, ABUND, TEMP, NOPH, IKEV, REGRID)
                LINEFLX (DEM, ABUND, EFFAR, WVLAR, TEMP, NOPH, IKEV, REGRID)

subroutines
        ARIA
        GRATFLX
        LINEFLX
```

```
history
        vinay kashyap (Sep98)
        bug fix for when FLSTR is ignored (VK; Oct98)
        added keyword ARSTR, added call to GRATFLX (VK; Nov98)
        converted to IDL5 notation (VK; OctMM)
        bug fix: crashed if FLX and WFLX set, but DWVL not set (VK;JanMMI)
        bug fix: FLSTR can have more than 8 tags (JJD; MayMMVII)
```

B.13 BREAK_RMF [/util]

```
procedure    break_rmf
       extract a sub-matrix from response matrix and  associated
       axes in OGIP compliant form. both input and output are given
       in a structure of the form:

        {NNRG, ELO, EHI, NCHAN, EMN, EMX, N_GRP, F_CHAN, N_CHAN, MATRIX, FIRSTCHAN}

        where ELO and EHI refer to range of photon energies at which the
        response is valid, EMN and EMX refer to bin boundaries for each
        channel, N_GRP refers to number of groups of non-zero data, F_CHAN
        refer to beginning indices of channels, N_CHAN are the number of
        channels in each group, and the output response MATRIX excludes
        zeros to save space.  The value of FIRSTCHAN indicates whether F_CHAN
        indices are 0-based or 1-based. SEE RD_OGIP_RMF().

        http://legacy.gsfc.nasa.gov/docs/heasarc/caldb/docs/memos/cal_gen_92_002/cal_gen
        for a description of the format

syntax
        break_rmf, rmfstr, erng, _extra=e

parameters
        rmfstr  [INPUT; required] response matrix structure of the form :
               {NNRG, ELO, EHI, NCHAN, EMN, EMX, N_GRP, F_CHAN, N_CHAN, MATRIX,
               FIRSTCHAN} as read in by e.g. RD_OGIP_RMF()
       erng    [INPUT; required] two element array designating the channel
               energy range to which to limit the output rmf

keywords
       effar    [OUTPUT] returns the effective area as a function of ELO
                * if an RMF file is read instead of an RSP file, EFFAR are
                uniformly 1, unless it is ASCA in which case it is the QEs

history
       liwei lin (Dec 2004)

       usage
k = ’ok’ & np=n_params() & nrng = n_elements(erng)
f np lt 2 then ok=’Insufficient parameters’ else $
 if nrng ne 2 then ok=’ERNG must be 2-element array’
f ok ne ’ok’ then begin
 print, ’Usage: break_rmf, rmfstr, erng’
 if np ne 0 then message,ok,/info
```

```
 return, -1L
ndif
f n_tags(rmfstr) eq 0 then return, -1L

       deconstruct rmf structure
NRG=rmfstr.NNRG & ELO=rmfstr.ELO & EHI=rmfstr.EHI
_GRP=rmfstr.N_GRP & F_CHAN=rmfstr.F_CHAN
_CHAN=rmfstr.N_CHAN & FIRSTCHAN=rmfstr.FIRSTCHAN
ATRIX=rmfstr.MATRIX & NCHAN=rmfstr.NCHAN
MN=rmfstr.EMN & EMX=rmfstr.EMX
zn=size(N_CHAN)

       set to zero all elements ouside erng
han = [emn+emx]/2
o   = where(chan lt erng(0) or chan gt erng(1),n)
oi  = where(chan ge erng(0) and chan lt erng(1),ni)
atrix[oo,*] = -1.0
matrix = fltarr(nchan-n,nnrg)  & nngrp = n_grp
f_chan = f_chan  & nn_chan = n_chan

       compress response matrix
or j = 0, nnrg-1 do begin
   ogipzip, matrix[*,j],carr,ngrp,fchan,nchano,eps=-0.01,chan0=firstchan,cchan=cchan
   nmatrix[*,j]=carr  & nngrp(j) = ngrp
   nf_chan(j) = fchan & nn_chan(j) = nchano
ndfor

rmfstr=create_struct('NNRG',nnrg,'ELO',elo,'EHI',ehi,$
                     'NCHAN',ni,'EMN',emn(ooi),'EMX',emx(ooi),$
                     'N_GRP',nngrp,'F_CHAN',nf_chan,'N_CHAN',nn_chan,$
                     'MATRIX',nmatrix,'FIRSTCHAN',firstchan)

      effar
   effar = fltarr(nnrg)
   for j = 0, nnrg-1 do effar(j) = total(nmatrix[*,j])

eturn, nrmfstr

nd
```

B.14 CALC_FTEST [/stat]

```
function        calc_ftest
        calculate the significance of a new model component using the
        F test.  Returns the p-value that the observed change in the
        chisq values for the given degrees of freedom can be obtained
        as a fluctuation.

syntax
        pval=calc_ftest(dof1,chi1,dof2,chi2,fstat=fstat,/wdchi, /double)

parameters
        dof1    [INPUT; required] degrees of freedom of SIMPLE model
        chi1    [INPUT; required] min chisq of SIMPLE model
        dof2    [INPUT; required] degrees of freedom of COMPLEX model
        chi2    [INPUT; required] min chisq of COMPLEX model
                * if arrays, the sizes of all of them must match.
                  the output will then be an array of the same size.
                * NOTE: complex model _must_ be a subset of simple model
                  and the parameter values that define one of the models
                  should not lie on the boundary of the other.

keywords
        fstat   [OUTPUT] the value of the F statistic
                * (CHI1/DOF1)/(CHI2/DOF2)
                * larger the FSTAT, the more plausible the complex model
        wdchi   [INPUT] if set, uses a different F statistic, defined as
                (DELTA_CHI/DELTA_DOF)/(CHI2/DOF2) =
                        ((CHI1-CHI2)/(DOF2-DOF1))/(CHI2/DOF2)
                * WARNING: if DOF1 is smaller than DOF2, that is, if
                  it appears that the input is already in this format,
                  additional subtractions are not done.

        _extra  [INPUT ONLY] pass defined keywords to IBETA()

description
        this is how CIAO/Sherpa's calc_ftest program works.
        I just implemented it as a quick way to get at the
        same results in IDL.

        Avoid using the F test unless you know that it is
        applicable for the problem you are working on.  See
        Protassov et al., 2002, ApJ, 571, 545
          http://adsabs.harvard.edu/abs/2002ApJ...571..545P
```

```
history
        vinay kashyap (MMXII.VII)
```

B.15 CAT_ID [/]

```
function        cat_id
        returns a concatenated structure of Line IDs

syntax
        C=cat_id(A,B,pick=pick,ask=ask,/okev,comm=comm,flst=flst,dbdir=dbdir,$
        fluxes=fluxes,fluxerr=fluxerr)

parameters
        A       [INPUT; required] output of LINEID
        B       [INPUT] output of LINEID to be merged with A
                * if any wavelengths are common, the IDs in A take
                  precedence (but see keyword ASK)
                * if not given, then the program *prints* a summary
                  of the IDs to the screen

keywords
        pick    [INPUT; default: all] position indices of elements in A
                to be selected (or not).
                * if scalar, returns all elements EXCEPT the specified one
                * has no effect on B
                * use these two to delete entries, for e.g.
        ask     [INPUT] if set, asks for confirmation before throwing
                away an ID from B that also exists in A.  allows overriding
                the default action.
                * ASK='n' will reverse the default by throwing away the
                  ID from A instead of the one from B
                * ASK='r' *reverses* the default, in that if there is a
                  component in B that's also in A, the one in A is >replaced<
                  by the one in B (as opposed to if ASK='n', then the one in
                  A is deleted, and B is just appended at the end)
                * ASK='k' overrides the default by keeping BOTH
        okev    [INPUT] if set, converts wavelengths [Ang] to energy [keV]
        comm    [INPUT] appends a comment to each line while printing
                * takes the comments from A.LABL(1,i)+A.LABL(0,i)
                * truncates the comment to a maximum of 30 characters
                  unless COMM is set to a higher number
                * for this to work when FLST is set, make sure DBDIR is
                  also defined
        flst    [INPUT] if set, prints out the line list in the form that
                RD_LIST likes
                * if FLST is a string, writes to file FLST
       mplet   [OUTPUT] if set, will contain on output a 2 dimensional MPLET
               string array as prescribed by MIXIE()
```

```
        dbdir   [INPUT] if set and FLST is set, appends DBDIR to IDs
                * 1 => $SCAR , 2 => $SPEX , 3 => $CHIANTI
        fluxes  [INPUT] if set and matches either the number of components
                or the number of matches, then renormalizes/overwrites the
                line fluxes in the output.
                * ONLY if IDs are being printed
        fluxerr [INPUT] if set and matches either the number of components
                or the number of matches, then appends an error to output
                * ignored if FLST is not set
                * if FLUXERR is given for each match, square-adds the errors
                  for the components
                * if single element, but there are more than 1 component to
                  ID structure, this is assumed to be the fractional error
                  (if < 1) or S/N (if > 1) for each match
        _extra  [JUNK] here only to prevent crashing the program

restrictions
        * works ONLY on the outputs of LINEID
        * requires subroutines:
          -- INICON
          -- CREATE_STRUCT
          -- LINEID

history
        vinay kashyap (Feb97)
        bug -- was breaking down in simplest case (VK; Mar97)
        was breaking down if ID was unknown (VK; Jul97)
        added keyword FLST (VK; Oct98)
        added call to INITSTUFF; added Tmax to output; added keywords DBDIR,
          FLUXES, FLUXERR; corrected bug re undefined output when FLST was set;
          increased number of significant digits in FLST output; generalized
          ASK to "Yes/No/Replace/Keep" (VK; Nov98)
        updated FLUXERR behavior to account for new field (VK; Dec98)
        modified to allow for field NOTES to be printed out (VK; Mar99)
        changed call to INITSTUFF to INICON (VK; 99May)
        added FLUXERR to output (VK; MarMM)
        converted to IDL5 array notation (VK; OctMM)
        changed keyword KEV to OKEV (VK; JanMMI)
        now /COMM also works when FLST (and DBDIR) are defined (VK;Nov01)
       added MPLET keyword (LL; Feb04)
        adjusted print format a bit to avoid **** in FLUXERR (VK; Jul12)
```

B.16 CAT_LN [/]

```
function        cat_ln
        returns a concatenated structure of Line emissivities

syntax
        C=cat_ln(A,B,pick=pick,ask=ask,comm=comm,reord=reord,/okev,$
        abund=abund,/noph,effar=effar,wvlar=wvlar,dem=dem,/temp,/ikev,/regrid)

parameters
        A       [INPUT; required] output of RD_LINE
        B       [INPUT] output of RD_LINE to be merged with A
                * if any wavelengths are common, the IDs in A take
                  precedence (but see keyword ASK)
                * if the temperature grids do not match, the emissivities
                  in B are interpolated to the same grid as in A
                * if not given, then the program *prints* a summary
                  of the lines to the screen

keywords
        pick    [INPUT; default: all] position indices of elements in A
                to be selected (or not).
                * if scalar, returns all elements EXCEPT the specified one
                * has no effect on B
                * use these two to delete entries, for e.g.
        ask     [INPUT] specifies how to handle duplicate lines
                * if not set, doesn't care about duplicates and keeps all
                * if set (see exceptions below), will discard duplicates
                  from B
                * ASK='n' will reverse the normal by throwing away the
                  line from A instead of the one from B
                * ASK='r' will >replace< the emissivity, etc. in A with
                  that of B
                * ASK='k' will act just as though ASK has not been set
        okeV    [INPUT] if set, converts wavelengths [Ang] to energy [keV]
                in the screen listing
        comm    [INPUT] appends a comment to each line while printing
                * takes the comments from A.DESIG[1,i]+A.DESIG[0,i]
                * truncates the comment to a maximum of 30 characters
                  unless COMM is set to a higher number
        flst    [INPUT] if set, prints out the line list in the form that
                RD_LIST likes
                * if FLST is a string, writes to file FLST
        reord   [INPUT] if set, rearrange the order of the output
                * +ve: descending order of fluxes
```

```
                * -ve: ascending order of fluxes
                * if abs(REORD) .NE. 1, limit output to first REORD entries
        _extra  [INPUT ONLY] use to pass defined keywords to
                LINEFLX: ABUND, NOPH, EFFAR, WVLAR, DEM, TEMP, REGRID

restrictions
        * works ONLY on the outputs of RD_LINE
        * requires subroutines:
          -- LINEFLX [GETABUND, WHEE]
          -- RD_LINE
          -- CREATE_STRUCT
          -- INICON

history
        vinay kashyap (Jun98, based on CAT_ID)
        changed keyword LNLST to FLST, added ABUND and REORD (VK; Jul98)
        delete keyword ABUND, and added call to LINEFLX (VK; Aug98)
        corrected bug that was skipping B because of roundoff error in
          first element; bug in handling ECONF (VK; Nov98)
        changed keyword KEV to OKEV to avoid conflict with LINEFLX keyword
          of same name (VK; 99Apr)
        added support for structure field JON (VK; 99May)
        allowed merging of mismatched LOGT grids (VK; 99Jul)
        converted to IDL5 array notation (VK; OctMM)
        added full RD_LIST compatible printout for FLST; changed output
          file wavelength format from f8.4 to f10.5 (VK; JanMMI)
        bug -- crashing on <empty> for FLST -- fixed (VK; Jul04)
        added extra columns (FLUX, logTMAX) to output of FLST (VK; Feb08)
```

B.17 CH_GET_FILE [/external]

```
PROJECT     : CHIANTI

      CHIANTI is an Atomic Database Package for Spectroscopic Diagnostics of
      Astrophysical Plasmas. It is a collaborative project involving the Naval
      Research Laboratory (USA), the University of Florence (Italy), the
      University of Cambridge and the Rutherford Appleton Laboratory (UK).


NAME        : CH_GET_FILE

PURPOSE     : to select a file from either a selected directory or the working
              directory, having an extension.


EXPLANATION : a file in  either a selected directory or the working
              directory, having an extension can be selected using a
              widget. Note that both directory and extension have to be
              supplied. If no file is found, an empty string is returned.

USE         : IDL> name = ch_get_file( ’˜/’, ’.pro’,  tit=’ Select a procedure ’)


EXAMPLES    : dir= concat_dir(!xuvtop),’dem’)
               dem_name=ch_get_file(path=dir,filter=’*.dem’,title=’Select DEM File’)


INPUTS      : directory, extension


OPT. INPUTS :


OUTPUTS     : the file name

OPT. OUTPUTS:


KEYWORDS    : title


CALLS       : findfile, break_file

COMMON      : co
```

```
RESTRICTIONS:  both directory and extension have to be
              supplied.


SIDE EFFECTS:

CATEGORY    :

PREV. HIST. : extracted from CDS/CHIANTI routines.



WRITTEN     :

      Giulio Del Zanna (GDZ),
       DAMTP  (University of Cambridge, UK)

MODIFIED    : Version 1, GDZ 10-Oct-2000
              V.2, GDZ, corrected a typo at the end of the file.
              V.3, GDZ,  generalized directory concatenation to work for
                  Unix, Windows  and VMS.

              V. 4, 19-July-2002, GDZ

               Added the option to select files also with the standard IDL
               dialaog_pickfile, and changed a few things...

              V.5, 2-Aug-02, GDZ
                reduced the size of the widget.

              V.6, 12-Aug-02, GDZ
                corrected for a bug in the directory output.

              V.7, 3-Nov-03  GDZ
                Fixed a bug when using Windows, the returned path was not
                correct.

VERSION     :  7,  3-Nov-03
```

B.18 CHANARF [/]

```
function        chanarf
        compute the average effective area over each channel, as a
        weighted average of the effective area from all the energies
        that contribute to that channel.

        makes a difference mainly for low and medium spectral
        resolution data.  for high-res grating data, not so much.

syntax
        carf=chanarf(rmfstr,effar,spec=spec)

parameters
        rmfstr  [INPUT; required] response matrix structure, in the
                same format as returned from RD_OGIP_RMF()
        effar   [INPUT] effective area as a function of energy [cm^2]
                * assumed to be on the same grid as the input energies
                  of RMFSTR, i.e., RMFSTR.ELO and RMFSTR.EHI
                * ignored if size doesn't match RMFSTR

keywords
        spec    [INPUT] a spectrum to further weight the contribution
                of EFFAR
                * assumed to be on the same grid as RMFSTR.ELO and RMFSTR.EHI
                * ignored if size doesn't match EFFAR
                * ideally should be in units of [ph/keV/...]
        chnrg   [OUTPUT] the average energy of a photon in this channel
                * it should be similar to 0.5*(RMFSTR.EMN+RMFSTR.EMX), and
                  serves as a check on the gain calibration
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing

history
        vinay kashyap (MMVI.IX)
```

B.19 CHANDRA_STATUS_BIT [/util]

```
function        chandra_status_bit
        filters a Chandra event file or status bit array on each status bit
        and reports how many events are flagged, and returns the indices of
        the filtered events

syntax
        idx=chandra_status_bit(inp,filter,bitstr=bitstr,verbose=verbose)

parameters
        inp     [INPUT; required] the input from which the ststus bits are
                to be filtered
                * if byte array, must be of size (4,NPH), and this is the
                  IDL read-in of a 32-bit string of 0s and 1s.  Note that
                  the columns are ordered last to first.
                * if a structure, there must be a field in it named .STATUS
                  and the byte array of above case is taken from that
                * if string, assumed to be the full path to a filename
                  that contains an events list, and it is read in and
                  the STATUS column is extracted from it
        filter  [INPUT; optional] the filter to be used, in the same
                format as CIAO.  e.g., HRC-I status-bit filtering uses
                'xxxxxx00xxxx0xxx00000000x0000000', i.e., 0 or 1 matches
                exactly, and 'x' ignores it.

keywords
        bitstr  [OUTPUT] a structure that contains the bits that are
                turned on, the number of events that are flagged, and
                for each of those, the array of the flagged indices
        stabit  [OUTPUT] ordered array of status bits, converted from
                IDL's 4xNPH to 32xNPH
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

requires
        B11001001
        IDL-Astro library

history
        vinay kashyap (2014jul, inspired by CIAO's summarize_status_bits)
```

B.20 CHIOMPRESS [/scrypt]

```
chiompress.pro
        a script to compress the CHIANTI databases into a smaller dataset
        to make it easy to transport and easy to read.  The max emissivity
        of each line is tested against the max emissivities of nearby lines
        and the line is retained iff it is above the locally defined threshold.

usage
        chifulldir='/data/fubar/SCAR/emissivity/chi713' ;full database dir
        outdir='/data/fubar/SCAR/emissivity/chiompress713'      ;compressed dir
        dynrng=1e2      ;dynamic range
        hwdth=1.0       ;half-width of window in AA
        .run chiompress

side effects
        creates files in OUTDIR
        creates a postscript file in $CWD

vinay kashyap (2010jul)
        bug fix: density sensitive lines were being ignored in the
        write (VK; 2010sep)
```

B.21 CHIPOACOM [/esempio]

```
script chipoacom

       Compares emissivities in PoA database to emissivities
       calculated in real time with CHIANTI.

        The CHIANTI calculation is done with CH_SYNTHETIC.PRO,
        a major component of CHIANTI's flagship spectral-synthesis
        program CH_SS.PRO.  CH_SYNTHETIC is used to calculate G(T),
        after which PoA's LINEFLX() is used to calculate intensities.
        These intensities are compared to intensities generated
        using PoA's RD_LINE(), FOLD_IONEQ(), and LINEFLX().
        Comparisons of the binned spectra are plotted.
        VOORSMOOTH() is used to smooth binned spectra.

syntax
        .run chipoacom

inputs
        !IONEQF ion-balance file
        !LDBDIR atomic database to use default =!LDBDIR
        !EDENS  electron density
        V_EM    emission measure components corresponding
                to each T [cm^5/logK]
                default is [6.1d11, 6.1d11, 7.1e11]
        V_LOGT  temperatures at which emission measures are defined
                default is [6.1, 6.8, 7.2]
        V_WMIN  minimum wavelength default = 5.0
        V_WMAX  maximum wavelength default = 15.0
        V_NWBIN number of bins in output spectra default = 8000
        V_Np    number of plots with which the resulting spectra
                are displayed. The wavelength range in each plot
                is determined by signal strength.
        V_SMOOT smoothing scale used to smooth binned spectra
outputs
        V_OUTPS name of ps output file

how to use
        1. initialize PINTofALE using INITALE
        2. parameters set with initale are:
           -- ATOMIC: !LDBDIR, !IONEQF, !CHIDIR
           -- STELLAR: !EDENS
        3. set the wavelength range of interest
              V_WMIN = 5.0
```

```
              V_WMAX = 180.0
        4. set the emission measures and corresponding temperatures
        5. set the output filename if output ps is desired
              V_OUTPS ='CHI4PoAcomparison.ps'
        6. run this script

history
        Liwei Lin (Jun03)
        bug fix noclip=0 and polyfill could result in 'inverse' polyfill
          (LL; Jul03)
        bug fix using floating point precission when defining wgrid
          is inadequate and results in incongruous specrtal binning
          results via hastogram(). use double precission instead (LL; Jul03)
        couple of bug fixes (VK; Apr04)
       add check for CHIANTI and IDL version compatiblity (LL; Dec05)
        updated for IDL5.6 keyword_set([0]) behavior change for vectors
          (VK; 20Mar2006)
       bug fix extraneous /2 after call to lineflx (LL; Aug08)
```

B.22 CLTSMOOTH [/util]

```
function        cltsmooth
        An adaptive smoothing technique that takes advantage
        of the central limit theorem.  Applicable to background
        subtracted light curves or spectra that need to have
        the negative values removed without touching anything
        that is at large S/N.

        First picks out all the -ves, assumes these are normally
        distributed, and collects all +ves that are of the same
        magnitude.  Then computes mean and stderr at all selected
        points, then at progressively smaller scales until at most
        32 bins widths, and resets the local mean and stderr iff
        it is significantly different from that computed at larger
        scale.

syntax
        sy=cltsmooth(y,syerr=syerr,swidth=swidth,verbose=verbose)

parameters
        y       [INPUT; required] the spectrum or light curve

keywords
        syerr   [OUTPUT] the errors computed at each bin
                * note that these will not be independent
        swidth  [OUTPUT] the widths over which the SY and
                SYERR are computed
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to avoid crashing the program

these are all the smoothing tools in PINTofALE
        ALALOESS() makes a loess curve
        CLTSMOOTH() removes -ves from background subtracted spectra or light curves
        CONV_RMF convolves with specified RMF
        HAARTRAN() smoothes by filtering on Haar wavelet coefficients
        LINEREM() removes lines from spectrum by iterative S/N filtering
        NOISMOOTH() does boxcar accumulation a la Ebeling's asmooth
        REGROUP() accumulates from one end
        SCRMF does fast convolution using optimized RMF
        SMOOTHIE does peak-descent accumulation to build up S/N
        SPLAC computes a segmented piecewise linear approximations
        UNKINK() removes sharp kinks from a curve
        VARSMOOTH() does local smoothing at specified scales
        VOORSMOOTH() does fast local smoothing at specified scales
```

`vinay kashyap (2015feb)`

B.23 COLBEHR [/stat]

```
function        colbehr
        computes hardness ratios based on 3-band data by invoking
        BEHR (Bayesian Estimate of Hardness Ratios) twice, and
        returns the inputs as well as the following quantities
        and their credible ranges in a structure:
          {S, M, H, lS=log10(S), lM=log10(M), lH=log10(H),
          SpM=S+M, MpH=M+H, SpH=S+H, T=S+M+H,
          SmM=S-M, MmH=M-H, SmH=S-H,
          R1=S/M, R2=M/H, R3=S/H,
          C1=C_SM, C2=C_MH, C3=C_SH,
          HR1=(S-M)/(S+M), HR2=(M-H)/(M+H), HR3=(S-H)/(S+H),
          HRA=(S-M)/(S+M+H), HRB=(M-H)/(S+M+H), HRC=(S-H)/(S+M+H)}

        Warning: Because of the necessity of combining two separate
        runs of BEHR, only the MCMC option is used.  Thus, if the
        length of the chain is small, the computed values may be
        subject to computational instability.

        Warning: BEHR assumes that the passbands in question
        do not overlap, and that the counts input to the program
        are statistically independent.  It is up to the users to
        ensure the validity of this assumption.  No checks are
        made to verify it.

        Reference:
        "Bayesian Estimation of Hardness Ratios: Modeling and Computations",
          Park, T., Kashyap, V.L., Siemiginowska, A., van Dyk, D., Zezas, A.,
          Heinke, C., and Wargelin, B., 2006, ApJ, 652, 610
        http://hea-www.harvard.edu/AstroStat/BEHR/

syntax
        behr3=colbehr(Ssrc,Msrc,Hsrc,Sbkg=Sbkg,Mbkg=Mbkg,Hbkg=Hbkg,$
        Sarea=Sarea,Marea=Marea,Harea=Harea,$
        Seff=Seff,Meff=Meff,Heff=Heff,$
        Sidx=Sidx,Midx=Midx,Hidx=Hidx,$
        Sscl=Sscl,Mscl=Mscl,Hscl=Hscl,$
        Stable=Stable,Mtable=Mtable,Htable=Htable,$
        /post,level=level,nsim=nsim,nburnin=nburnin,hpd=hpd,$
        outputf=outputf,BEHRdir=BEHRdir, verbose=verbose)

parameters
        Ssrc    [INPUT; required] counts in source region in the soft (S) band
        Msrc    [INPUT; required] counts in source region in the medium (M) band
```

```
        Hsrc    [INPUT; required] counts in source region in the hard (H) band
                * can be arrays; if so, the array with the most elements
                  determines the size of the output and the shortfalls in the
                  others, if any, are made up by replicating the first elements

keywords
        Sbkg    [INPUT] counts in background region in the S band
        Mbkg    [INPUT] counts in background region in the M band
        Hbkg    [INPUT] counts in background region in the H band
                * if not given, assumed to be 0
                * if size smaller than Xsrc, first element gets replicated
        Sarea   [INPUT] background scaling factor in the S band
        Marea   [INPUT] background scaling factor in the M band
        Harea   [INPUT] background scaling factor in the H band
                * (background region area)/(source region area)
                * if not given, assumed to be 1
                * can also include differences in exposure time into
                  the ratio, in the same manner as geometric area
                * if size smaller than Xsrc, first element gets replicated
        Seff    [INPUT] effective area in S band
        Meff    [INPUT] effective area in M band
        Heff    [INPUT] effective area in H band
                * if none are set, all are assumed to be 1,
                  else if one is set, all are assumed to be equal to that one,
                  else if two are set and unequal, third is assumed to be 1
                * can also be the effective area relative to some
                  special point on the detector (e.g., aimpoint)
                  or even some specific detector (e.g., ACIS-I v/s ACIS-S)
                * if size smaller than Xsrc, first element gets replicated
        Sidx    [INPUT] index of prior on S (range = 0+)
        Midx    [INPUT] index of prior on M (range = 0+)
        Hidx    [INPUT] index of prior on H (range = 0+)
                * if none are set, all are assumed to be 0.5,
                  else if one is set, all are assumed to be equal to that one
                  else if two are set and are unequal, third is assumed to be 0.5
                * if size smaller than Xsrc, first element gets replicated
                * similar to AGAMMA of PPD_SRC()
        Sscl    [INPUT] scaling index of prior on Ssrc
        Mscl    [INPUT] scaling index of prior on Msrc
        Hscl    [INPUT] scaling index of prior on Hsrc
                * if none are set, all are assumed to be 0
                  else if one is set, all are assumed to be equal to that one
                  else if two are set and are unequal, third is assumed to be 0
                * if size smaller than Xsrc, first element gets replicated
                * similar to BGAMMA of PPD_SRC()
```

```
        Stable  [INPUT] filename containing a tabulated prior for Ssrc
        Mtable  [INPUT] filename containing a tabulated prior for Msrc
        Htable  [INPUT] filename containing a tabulated prior for Hsrc
                * the table prior must be an ascii file with the following format:
                  line 1: number of entries, say NLIN
                  line 2: labels for the columns, ignored
                  lines 3..NLIN+2: two whitespace separated columns of numbers,
                    with each row containing the source intensity and the posterior
                    density, in that order
                * the default filenames are "./tblprior_{soft|med|hard}.txt"
                * the default filenames are used iff Stable, Mtable, and Htable are set
                  but are not found
                * WARNING: if regex is used in the filename specification, only the
                  first file from the list will be used.  furthermore, if specified,
                  the table priors are applied to _all_ SSRC, MSRC, and HSRC
        post    [INPUT] if set, suggests the values of (Sidx,Sscl), (Midx,Mscl),
                and (Hidx,Hscl) going forward, i.e., what you should set the
                priors to in your next calculation for the same source -- the
                suggested values are stored in the output structure
        level   [INPUT] percentage confidence level at which to report error
                (default = 68)
        details [INPUT] compute various ratios (true/false)?
                (default = true)
        nsim    [INPUT] number of draws if algo=gibbs (default=10000)
        nburnin [INPUT] number of burn-in draws if algo=gibbs
                (default=5000 or NSIM/2, whichever is smaller)
        outputF [INPUT] root of filename in which to place output
                (default = 'none')
                * output will be placed in the files OUTPUTF.txt and OUTPUTF_draws.txt
                * NOTE: if OUTPUTF='none', then MC draws will be in BEHR_draws.txt
        BEHRdir [INPUT] full path to directory where BEHR executable resides
                (default = '/data/fubar/kashyap/AstroStat/BEHR')
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

requirements
        uses subroutines HIPD_INTERVAL() and MODALPOINT()
        BEHR executable must be installed in BEHRDIR
        BEHR should be executable under the shell via SPAWN
        BEHR output assumed to be compatible with 12-19-2005 version

side-effects
        potentially creates numerous ascii files in $cwd or `basedir OUTPUTF`

history
```

```
        vinay kashyap (Mar07; based on behr_hug.pro)
        bug correction with NaNs not being caught in some cases
          (VK; Mar07)
        added keywords Stable,Mtable,Htable (VK; Feb08)

etymology
        getting color-color diagrams using BEHR
        (that's my story and I'm sticking to it)
```

B.24 COLOURCURVE [/]

```
function        colourcurve
        computes color (log(S)-log(H)) for given NH, for various temperatures,
        for a given ARF/RMF combo

syntax
        colcurv=colourcurve(NH,arffil,rmffil,logT,Sband=Sband,Hband=Hband,$
        n_e=n_e,logP=logP,abund=abund,$
        ldbdir=ldbdir,ceroot=ceroot,cdbdir=cdbdir,ioneqf=ioneqf,$
        lstr=lstr,cstr=cstr,verbose=verbose,$
        chidir=chidir, cocofil=cocofil,chdwvl=chdwvl,/twoph,/noff,/nofb,$
        nhne=nhne, fH2=fH2,He1=He1,HeII=HeII,/Fano,/wam,/bam,/mam,/vion,/noHeH,$
        icrstr=icrstr,ionfrac=ionfrac, vfkydir=vfkydir,,nconsec=nconsec,$
        EBV=EBV,R_V=R_V,LMC2=LMC2,AVGLMC=AVGLMC,ExtCurve=ExtCurve,$
        fmgamma=fmgamma,fmx0=fmx0,fmc1=fmc1,fmc2=fmc2,fmc3=fmc3,fmc4=fmc4)

parameters
        NH      [INPUT; required] H I column density in [cm^-2]
        arffil  [INPUT; required] name of ARF file
                * if blank, ignores RMFFIL and proceeds to compute
                  flux ratios rather than counts ratios
                  - hardcoded range of [15 kev, 200 Ang], edit the code, or
                    keep SBAND and HBAND in this range
        rmffil  [INPUT; required] name of RMF file
                * quits if ARFFIL and RMFFIL are set but files cannot be read
        logT    [INPUT] temperatures at which to compute colors [log10([K])]
                * if not given, or if input is not understandable, assumed
                  to be 5:8:0.1

keywords
        Sband   [INPUT] 2-element float array of energies for the soft band [keV]
                * default is [0.5,2]
                * if 1-element, then alters the nearest bound.  e.g., if
                  Sband=0.3, changes to [0.3,2]
                  Sband=1.24, changes to [1.24,2]
                  Sband=1.26, changes to [0.5,1.26]
                  Sband=4, changes to [0.5,4]
                * if integer, assumed to be channel number (cf. RMFFIL)
        Hband   [INPUT] 2-element float array of energies for the hard band [keV]
                * default is [2.0,7.0]
                * if 1-element, then alters the nearest bound (cf. SBAND)
                * if integer, assumed to be channel number (cf. RMFFIL)
        n_e     [INPUT] passed w/o comment to RD_LINE() and RD_CONT()
        logP    [INPUT] passed w/o comment to RD_LINE() and RD_CONT()
```

```
abund   [INPUT] abundances
        * default is to use solar coronal
ldbdir  [INPUT] line emissivity database
        * default is '$APED'
ceroot  [INPUT] what type of continuum to use
        * default is 'apec'
        * options are 'cie' and 'chianti'
          (but latter has problems; still being investigated)
cdbdir  [INPUT] continuum emissivity database
        * default is '$CONT'
ioneqf  [INPUT] ion balance file
        * default is 'ioneq/chianti.ioneq'
        * not used for APED/ATOMDB
lstr    [I/O] structure that contains line emissivity database
cstr    [I/O] structure that contains continuum emissivity database
        * if LSTR and CSTR are given on input, then the emissivity
          databases are not read in
verbose [INPUT] controls chatter
_extra  [INPUT ONLY] pass defined keywords to subroutines
        [RD_IONEQ] CHIDIR
        [RD_CONT] COCOFIL, CHDWVL, TWOPH, NOFF, NOFB
        [ISMTAU] FH2, HE1, HEII, FANO, VION, BAM, MAM, NOHEH, ICRSTR
        [IONABS] ICRSTR, IONFRAC, VFKYDIR, NCONSEC
        [FM_UNRED] EBV, R_V, LMC2, AVGLMC, EXTCURVE, FMGAMMA, FMX0,
                   FMC1, FMC2, FMC3, FMC4
        [LINEFLX] NHNE
```

B.25 CONV_RMF [/util]

```
procedure       conv_rmf
        convolves input energy spectrum with the response matrix

        unlike the combination of RDRESP and FOLD_RESP, this one goes
        easy on memory at the expense of speed.

syntax
        conv_rmf,nrg,flx,chan,spec,rmf,effar=effar,nrgar=nrgar,$
        rmfstr=rmfstr,fchcol=fchcol,/shift1,verbose=verbose

parameters
        nrg     [INPUT; required] mid-bin values at which input FLX is
                defined.
                * usually [keV], depends on RMF file
                * if size = N(FLX)+1, assumed to be bin-boundaries
        flx     [INPUT; required] fluxes in [ph/bin/(s/cm^2/sr/...)]
        chan    [OUTPUT; required] bin boundary values at which output SPEC
                is defined
                * same units as NRG
                * WARNING: the values that populate this array are taken
                  from the EMN and EMX values in the RMF.  These are not
                  exact values, and simply represent approximate bounds
                  on the detector channels that the input spectrum gets
                  redistributed into.  They are to be used mainly for
                  plotting purposes, and should not be used for scientific
                  purposes unless the user recognizes their limitations.
                  In other words, the array indices of SPEC are the
                  detector channels, and the CHAN array is a mapping from
                  channels to energy, as defined in the EBOUNDS extension
                  of the RMF
        spec    [OUTPUT; required] output, convolved spectrum
        rmf     [INPUT; required] name of OGIP-compliant FITS file containing
                the response matrix
                * may also be a structure, such as the output of RD_OGIP_RMF()

keywords
        effar   [INPUT] effective areas [cm^2]
        nrgar   [INPUT] energies at which EFFAR are defined
                * same units as NRG
                * sizes of EFFAR and NRGAR must match
                * range of NRGAR must overlap NRG
                * if EFFAR and NRGAR are legal, FLX is multiplied by
                  EFFAR (over intersection of NRGAR and NRG; zeroed
```

```
                  elsewhere) before convolving with response matrix
                * of course, this multiplication makes sense only if
                  units on FLX are [ph/bin/cm^2/...]
        rmfstr  [OUTPUT] structure containing info of response matrix,
                the output of RD_OGIP_RMF()
        verbose [INPUT] controls chatter
        _extra  [INPUT ONLY] pass defined keywords to subroutines:
                RD_OGIP_RMF: FCHCOL, SHIFT1
                REBINW : SLOWOK

restrictions
        requires IDLAstro library
        requires RD_OGIP_RMF()
        requires REBINW()
        requires BINERSP()

these are all the smoothing tools in PINTofALE
        ALALOESS() makes a loess curve
        CLTSMOOTH() removes -ves from background subtracted spectra or light curves
        CONV_RMF convolves with specified RMF
        HAARTRAN() smoothes by filtering on Haar wavelet coefficients
        LINEREM() removes lines from spectrum by iterative S/N filtering
        NOISMOOTH() does boxcar accumulation a la Ebeling's asmooth
        REGROUP() accumulates from one end
        SCRMF does fast convolution using optimized RMF
        SMOOTHIE does peak-descent accumulation to build up S/N
        SPLAC computes a segmented piecewise linear approximations
        UNKINK() removes sharp kinks from a curve
        VARSMOOTH() does local smoothing at specified scales
        VOORSMOOTH() does fast local smoothing at specified scales

history
        vinay kashyap (Apr99)
        added keyword SHIFT1 (VK; JanMMI)
        bug: incorrect usage of WVLAR instead of NRGAR in effar block
          (Erica R; Aug01)
        deleted keyword SHIFT1, cleaned up and speeded up (VK; Nov2001)
        bug: when spectrum size < RMF size and RMF has multiple groups,
          was using only the first group (VK; Nov'02)
        bug: was crashing when NRG was outside the bounds of the RMF
          in cases where the RMF was at a higher resolution (LL/VK; Apr'03)
        added keyword VERBOSE (VK; MarMMV)
        bug: when input spectrum energy range was smaller than RMF range,
          was assuming RMF was at higher resolution even if it wasn't
          (VK; Aug'07)
```

```
bug: wasn't checking for frequency beating between input spectrum
  grid and input RMF grid in above case; now if it does, and calls
  REBINW() and forces input spectrum to the right grid (VK; Aug'07)
bug: EFFAR multiplication was being ignored when input spectrum had
  to be rebinned to match RMF grid (VK; Oct'07)
```

B.26 CONVERT_TO_TYPE [/external]

```
NAME:
      CONVERT_TO_TYPE

PURPOSE:

      Converts its input argument to a specified data type.

AUTHOR:

      FANNING SOFTWARE CONSULTING
      David Fanning, Ph.D.
      1645 Sheely Drive
      Fort Collins, CO 80526 USA
      Phone: 970-221-0438
      E-mail: davidf@dfanning.com
      Coyote's Guide to IDL Programming: http://www.dfanning.com

CATEGORY:

      Utilities

CALLING SEQUENCE:

      result = Convert_To_Type(input, type)

INPUT_PARAMETERS:

      input:          The input data to be converted.
      type:           The data type. Accepts values as given by Size(var, /TNAME) or Si

OUTPUT_PARAMETERS:

     result:          The input data is converted to specified data type.

KEYWORDS:

    None.

RESTRICTIONS:

    Data types STRUCT, POINTER, and OBJREF are not allowed.

MODIFICATION HISTORY:
```

```
Written by David W. Fanning, 19 February 2006.
```

B.27 CONVERTNAME [/external]

```
PROJECT:  CHIANTI

PROJECT:  CHIANTI

      CHIANTI is an atomic database package for the calculation of
      continuum and emission line spectra from astrophysical plasmas. It is a
      collaborative project involving the Naval Research Laboratory
      (Washington DC, USA), the Arcetri Observatory (Firenze, Italy), and the
      Cambridge University (United Kingdom).

NAME:
       convertname

PURPOSE:
       Ion names as character strings are converted into
       numerical values (note c_2 is C II or C^+1
       in spectroscopic or atomic notation)

CATEGORY:

       naming utility

CALLING SEQUENCE:

      CONVERTNAME,Name,Iz,Ion


INPUTS:
       Name:   such as 'c_2'


OUTPUTS:

       Iz:  nuclear charge Z  (6 for 'c_2', the equivalent of C II)
      Ion:  ionization stage:  (2 for 'c_2')

OPTIONAL OUTPUTS

      DIELECTRONIC   Set to 1 if NAME has a 'd' appended to it
                     (indicating dielectronic recombination data) else
                     set to 0
```

```
EXAMPLE:

                    > convertname,'c_2',iz,ion
                    > print,iz,ion
                    > 6,2

                    > convertname,'o_6d',iz,ion
                    > print,iz,ion
                    > 8,6

MODIFICATION HISTORY:
       Written by:     Ken Dere
       March 1996:     Version 2.0
      October 1999:   Version 3.  by kpd

      Ver.4, 11-Dec-01, Peter Young
          Revised routine, removing ch_repstr call.
          Added DIELECTRONIC optional output.
```

B.28 CURVE_FIT [/external]

```
NAME:
      CURVEFIT

PURPOSE:
      Non-linear least squares fit to a function of an arbitrary
      number of parameters.  The function may be any non-linear
      function.  If available, partial derivatives can be calculated by
      the user function, else this routine will estimate partial derivatives
      with a forward difference approximation.

CATEGORY:
      E2 - Curve and Surface Fitting.

CALLING SEQUENCE:
      Result = CURVE_FIT(X, Y, Weights, A, SIGMA, FUNCTION_NAME = name, $
                        ITMAX=ITMAX, ITER=ITER, TOL=TOL, /NODERIVATIVE)

INPUTS:
      X:  A row vector of independent variables.  This routine does
          not manipulate or use values in X, it simply passes X
          to the user-written function.

      Y:  A row vector containing the dependent variable.

 Weights:  A row vector of weights, the same length as Y.
           For no weighting,
                Weights(i) = 1.0.
           For instrumental (Gaussian) weighting,
                Weights(i)=1.0/sigma(i)^2
           For statistical (Poisson)  weighting,
                Weights(i) = 1.0/y(i), etc.

      A:  A vector, with as many elements as the number of terms, that
          contains the initial estimate for each parameter.  IF A is double-
          precision, calculations are performed in double precision,
          otherwise they are performed in single precision. Fitted parameters
          are returned in A.

KEYWORDS:
      FUNCTION_NAME:  The name of the function (actually, a procedure) to
      fit.  IF omitted, "FUNCT" is used. The procedure must be written as
      described under RESTRICTIONS, below.
```

```
       ITMAX:  Maximum number of iterations. Default = 20.
       ITER:   The actual number of iterations which were performed
       TOL:    The convergence tolerance. The routine returns when the
               relative decrease in chi-squared is less than TOL in an
               interation. Default = 1.e-3.
       CHI2:   The value of chi-squared on exit (obselete)

       CHISQ:   The value of reduced chi-squared on exit
       NODERIVATIVE:   IF this keyword is set THEN the user procedure will not
               be requested to provide partial derivatives. The partial
               derivatives will be estimated in CURVEFIT using forward
               differences. IF analytical derivatives are available they
               should always be used.

OUTPUTS:
       Returns a vector of calculated values.
       A:  A vector of parameters containing fit.

OPTIONAL OUTPUT PARAMETERS:
       Sigma:  A vector of standard deviations for the parameters in A.

COMMON BLOCKS:
       NONE.

SIDE EFFECTS:
       None.

RESTRICTIONS:
       The function to be fit must be defined and called FUNCT,
       unless the FUNCTION_NAME keyword is supplied.  This function,
       (actually written as a procedure) must accept values of
       X (the independent variable), and A (the fitted function's
       parameter values), and return F (the function's value at
       X), and PDER (a 2D array of partial derivatives).
       For an example, see FUNCT in the IDL User's Libaray.
       A call to FUNCT is entered as:
       FUNCT, X, A, F, PDER
where:
       X = Variable passed into CURVEFIT.  It is the job of the user-written
           function to interpret this variable.
       A = Vector of NTERMS function parameters, input.
       F = Vector of NPOINT values of function, y(i) = funct(x), output.
       PDER = Array, (NPOINT, NTERMS), of partial derivatives of funct.
               PDER(I,J) = DErivative of function at ith point with
               respect to jth parameter.  Optional output parameter.
```

```
              PDER should not be calculated IF the parameter is not
              supplied in call. IF the /NODERIVATIVE keyword is set in the
              call to CURVEFIT THEN the user routine will never need to
              calculate PDER.

PROCEDURE:
      Copied from "CURFIT", least squares fit to a non-linear
      function, pages 237-239, Bevington, Data Reduction and Error
      Analysis for the Physical Sciences.  This is adapted from:
      Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear
      Parameters", J. Soc. Ind. Appl. Math., Vol 11, no. 2, pp. 431-441,
      June, 1963.

      "This method is the Gradient-expansion algorithm which
      combines the best features of the gradient search with
      the method of linearizing the fitting function."

      Iterations are performed until the chi square changes by
      only TOL or until ITMAX iterations have been performed.

      The initial guess of the parameter values should be
      as close to the actual values as possible or the solution
      may not converge.

EXAMPLE:  Fit a function of the form f(x) = a * exp(b*x) + c to
          sample pairs contained in x and y.
          In this example, a=a(0), b=a(1) and c=a(2).
          The partials are easily computed symbolicaly:
          df/da = exp(b*x), df/db = a * x * exp(b*x), and df/dc = 1.0

          Here is the user-written procedure to return F(x) and
          the partials, given x:

      pro gfunct, x, a, f, pder      ; Function + partials
        bx = exp(a(1) * x)
        f= a(0) * bx + a(2)         ;Evaluate the function
        IF N_PARAMS() ge 4 THEN $   ;Return partials?
        pder= [[bx], [a(0) * x * bx], [replicate(1.0, N_ELEMENTS(f))]]
      end

        x=findgen(10)                  ;Define indep & dep variables.
        y=[12.0, 11.0,10.2,9.4,8.7,8.1,7.5,6.9,6.5,6.1]
        Weights=1.0/y            ;Weights
        a=[10.0,-0.1,2.0]        ;Initial guess
        yfit=curvefit(x,y,Weights,a,sigma,function_name='gfunct')
```

```
        print, 'Function parameters: ', a
        print, yfit
      end

MODIFICATION HISTORY:
      Written, DMS, RSI, September, 1982.
      Does not iterate IF the first guess is good.  DMS, Oct, 1990.
      Added CALL_PROCEDURE to make the function's name a parameter.
             (Nov 1990)
      12/14/92 - modified to reflect the changes in the 1991
           edition of Bevington (eq. II-27) (jiy-suggested by CreaSo)
      Mark Rivers, U of Chicago, Feb. 12, 1995
          - Added following keywords: ITMAX, ITER, TOL, CHI2, NODERIVATIVE
            These make the routine much more generally useful.
          - Removed Oct. 1990 modification so the routine does one iteration
            even IF first guess is good. Required to get meaningful output
            for errors.
          - Added forward difference derivative calculations required for
            NODERIVATIVE keyword.
          - Fixed a bug: PDER was passed to user's procedure on first call,
            but was not defined. Thus, user's procedure might not calculate
            it, but the result was THEN used.

     Steve Penton, RSI, June 1996.
           - Changed SIGMAA to SIGMA to be consistant with other fitting
             routines.
           - Changed CHI2 to CHISQ to be consistant with other fitting
             routines.
           - Changed W to Weights to be consistant with other fitting
             routines.
           _ Updated docs regarding weighing.

       Vinay Kashyap, CfA, Dec 1998
             - Changed name to CURVE_FIT.
             - Added keyword _EXTRA=E to calling sequence and to all
               calls to CALL_PROCEDURE.
```

B.29 CURVESECT [/util]

```
function        curvesect
        finds and returns the point(s) of intersection of two curves

syntax
        xy=curvesect(y1,y2,xpt,verbose=verbose)

parameters
        y1      [INPUT; required] points of curve Y1(X)
        y2      [INPUT; required] points of curve Y2(X)
                * Y1 and Y2 must be defined on the same grid
        xpt     [INPUT] x-coordinates of Y1 and Y2
                * if not supplied, uses the array index

keywords
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

restrictions
        only returns the intersections within the domain of XPT
        inputs must be sorted in X

description
        a very simple algorithm that checks whether two curves
        have any crossings, and if they do, finds the point of
        intersection using the line segments that connect across
        the crossings.  Don't push it.

example
        .run curvesect

history
        Vinay Kashyap (2012dec)
```

B.30 CUT_ID [/]

```
function        cut_id
        add or delete IDs to an existing ID structure

syntax
        newid=cut_id(idstr,idx,delet=delet,addon=addon,verbose=verbose,$
        /incieq,dbdir=dbdir,sep=sep,pres=pres,logP=logP,n_e=n_e,$
        chifil=chifil,chidir=chidir,eqfile=eqfile)

warning
        will not keep track of relative fluxes or their errors properly,
        though the total flux will be conserved.

parameters
        idstr   [INPUT; required] an ID structure (see LINEID.PRO) containing
                line identifications of spectral features
        idx     [INPUT; required] a zero-based index of the feature
                being (re)edited
                * must be a scalar
                * nothing happens if IDX is outside the legal range defined
                  by IDSTR

keywords
        delet   [INPUT] set to a zero-based index to IDSTR.(IDX+1)
                * if float, assumed to refer to IDSTR.(IDX+1).WVL
                * if string, assumed to be in same format as understood by
                  RD_LIST: "Z ION <sep> WAVE <sep> SOURCE <sep> DESCRIPTION"
                  (RD_LIST will in fact be called in order to decipher it)
                * may be an array
        addon   [INPUT] string describing the line to be added as an ID
                to IDSTR.(IDX+1)
                * must be in same format as that understood by RD_LIST:
                  "Z ION <sep> WAVE <sep> SOURCE <sep> DESCRIPTION"
                * may be an array
        eps     [INPUT] a small number
                * default is 1e-5
        verbose [INPUT] controls chatter
        _extra  [INPUT ONLY] pass defined keywords to
                RD_LIST: INCIEQ,DBDIR,SEP,PRES,LOGP,N_E,CHIFIL,CHIDIR,EQFILE
                LINEFLX: DEM,ABUND,NOPH,EFFAR,WVLAR

restrictions
        requires subroutines:
          RD_LIST, RD_LINE, FOLD_IONEQ, READ_IONEQ, RD_IONEQ, LINEFLX,
```

```
          GETABUND, WHEE, SYMB2ZION, LAT2ARAB, CAT_LN
        requires IDL 5.3+ (because of use of STRSPLIT, STRJOIN, STRCMP)

history
        vinay kashyap (JanMMI)
```

B.31 DATA_CHK [/external]

```
Purpose: checks input data for type, ndimension, etc
          (uses IDL size function results)

Keyword Parameters:
   type -   if set, return idl data type  from size function
             type            data type
               0               ???
               1               byte
               2               integer
               3               longword
               4               floating point
               5               double precision
               6               ???
               7               string
               8               structure
   ndimen - if set, return number dimensions (size(0))
   nimages - if set, return #images (1->2D, NI->3D, else 0)
   xsize - if set, return X size (size(data))(1)
   nx    - synonym for xsize
   ysize - if set, return Y size (size(data))(2)
   ny    - synonym for ysize
   string/struct/undefined - if set, return true if type matches
   scalar, vector - if set, true if data of specified variety

Calling Examples:
   if (data_chk(p1,/type) eq data_chk(p2,/type)) then...
   case data_chk(data,/type) of...
   if data_chk(data,/string,/scalar) then ...
   if data_chk(data,/string,/struct,/undef,/orr)
   case data_chk(maybe_cube,/nimages) of...

History:
   27-Apr-1993 (SLF)
   21-Mar-1994 (SLF) documentation header
   10-oct-1996 (SLF) add SCALAR (synonym for the historical mispell SCALER)
    2-oct-1997 (SLF) add NIMAGES
   15-nov-1997 (SLF) add XSIZE and YSIZE keyword and function
   20-oct-2002 (LWA) added list of data types to header
    2-jan-2007 (SLF) removed vestigal references to ORR (not supported)

Restrictions:
   some keywords are mutually exclusive - for self-documenting code
```

```
and reduction of code duplicataion
ORR not supported (place holder never realized - you'll have to
do your own 'ands' and 'ors' external (could write data_chks.pro
driver with keyword inherit via _extra to This routine...)
```

B.32 DATE2DECIMAL [/util]

```
function        date2decimal
                compute and return decimal representation of date
                as fraction of the year.

syntax
        date=date2decimal(datestr,fod=fod,doy=doy,jday=jday)

parameters
        datestr [INPUT; required] date string in format 'YYYY-MM-DDTHH:MM:SS'
                * can be an array
                * if given in an ununderstandable format, assumed to be today, now.

keywords
        fod     [OUTPUT] fraction of day
        doy     [OUTPUT] day of year
        jday    [OUTPUT] Julian day

history
        vinay kashyap (2014nov)
        bug fix: Feb was resetting doy counter (VK; 2015jun)
```

B.33 DATE2TIMESTAMP [/util]

```
function        date2timestamp
        given a decimal date (in units of [yr]), returns the calendar
        date as a timestamp in the format YYYY-MM-DDTHH:MM:SS.S

syntax
        tstamp=date2timestamp(date,year=year)

parameters
        date    [INPUT; required] decimal date in years or fraction
                of year (see keyword YEAR)

keywords
        year    [INPUT] if given, adds this to DATE
                * useful for when only fraction of year is given
        _extra  [JUNK] here only to prevent crashing the program

example
        .run date2timestamp
        (calls date2decimal)

history
        Vinay Kashyap (2015.4863)
```

B.34 DEG2HRS [/util]

```
function        deg2hrs
        converts decimal degrees to sexagesimal system and returns a
        string array of properly formatted output

parameters
        d       [INPUT; required] degrees
                * may even be a string of comma separated numbers
        h       [OUTPUT] hours
        m       [OUTPUT] minutes
        s       [OUTPUT] seconds

keywords
        dec     [INPUT] if set, does not divide by 15 (the RA v/s Dec thing)
        _extra  [INPUT] allows passing defined keywords to
                STR_2_ARR: SEP, R8, SQUISH

subroutines
        STR_2_ARR
        see also HRS2DEG

history
        vinay kashyap (Nov98)
        changed call to str2arr to str_2_arr (VK; Sep14)
```

B.35 DEMACS [/]

```
function        demacs
        a DEM editor that allows interactively "tweaking" a specified DEM

syntax
        dem=demacs(logt,dem0=dem0,logt0=logt0,norm=norm,slope=slope,$
        group=group,igroup=igroup, PLOT_KEYWORDS)

parameters
        logt    [INPUT; required] log10(Temperature [K]) at which DEM
                is to be determined

keywords
        dem0    [INPUT] initial DEM (just for guidance).
                * if size does not match LOGT0, gets stretched to suit.
                * overrides SLOPE, but not NORM
                * if max(DEM0)<100, assumed to be in log form -- i.e.,
                  plots are in "linear" form and +vity is not enforced.
        logt0   [INPUT] logT at which DEM0 are defined.  if not given,
                assumed to be LOGT.
        norm    [INPUT] if set, first element of initial DEM is set to NORM
        slope   [INPUT; default=1] if given, generates initial
                DEM=NORM*T0^(SLOPE)
                * if DEM0 is specified, SLOPE is ignored
        group   [OUTPUT] long-int array of same size as LOGT containing
                grouping information
        igroup  [OUTPUT] array of same size as GROUP, containing index
                of "representative element" of each group
        _extra  [INPUT] allows passing defined keywords (e.g., YRANGE)
                to PLOT.

restrictions
        requires X-windows, display, and mouse capability.

history
        vinay kashyap (Feb97)
        corrected log behavior, added _extra to PLOT (VK; Apr97)
        cleaned up log behavior, added keystrokes *,/,v,^,z (VK; Aug98)
        button press status now stored in !MOUSE, not !ERR (VK; Apr09)
```

B.36 DESCALE_ALL [/external]

```
EXPLANATION

     This routine descales all types of spline fits into upsilons or
     rates, i.e., it does both electron upsilons and proton rates,
     and both 5-point and 10-point splines. In addition it can
     simultaneously descale several temperatures at once.

INPUTS

     TEMP     Temperature(s), K.

     SPLSTR   Structure output by read_splups.

     INDEX    Index of structure.

OUTPUTS

     UPS      Upsilon value(s) at temperature(s) TEMP.

EXAMPLES

     read_splups,splupsfile,splstr
     descale_all,[1.e6,2.e6],splstr,5,ups
     print,ups

HISTORY

     Ver.1, 15-Mar-01, Peter Young
              adapted from Ken Dere's descale_ups.pro.

     Ver.2, 12-Nov-01, Peter Young
              added type 6 transitions (for protons)

     Ver.3, 3-Jan-2012, Ken Dere
              allowed for the number of spline points from 5 through 9
```

B.37 DESCALE_UPS_VK [/external]

```
PROJECT:  CHIANTI

      CHIANTI is an atomic database package for the calculation of
      continuum and emission line spectra from astrophysical plasmas. It is a
      collaborative project involving the Naval Research Laboratory
      (Washington DC, USA), the Arcetri Observatory (Firenze, Italy), and the
      Cambridge University (United Kingdom).


NAME:
       DESCALE_UPS_VK
       (used to be) DESCALE_UPS

PURPOSE:

       convert from Burgess-Tully scaling spline fits to Upsilons

CATEGORY:

       science.

CALLING SEQUENCE:

      DESCALE_UPS,Index,Jndex,xt,upsilion, T_TYPE,C_UPS,SPLUPS

INPUTS:

       Index:  index of lower energy level (lowest level is 1)
       Jndex:  index of upper energy level (lowest level is 1)
       xt:  scaled temperature

       T_TYPE: 2 dimensional array contain values of the transition type
       C_UPS: 2 dimensional array containing values of the Burgess and Tully
               scaling parameter c
       SPLUPS: spline fits to the scaled Upsilons

OPTIONAL INPUTS:

       None:

KEYWORD PARAMETERS:
```

```
       None:

OUTPUTS:

       Upsilon:  the Maxwellian averaged collision strength



COMMON BLOCKS:

       NONE:

        ;common elvlc,l1a,term,conf,ss,ll,jj,ecm,eryd,ecmth,erydth,eref
      ;common wgfa, wvl,gf,a_value
      ;common upsilon,t_type,c_ups,splups


PROCEDURE:

       see Burgess and Tully, 1992, Astron and Astrophys, 254, 436.

EXAMPLE:

            ;
MODIFICATION HISTORY:
       Written by:     Ken Dere
       March 1996:     Version 2.0
      December 1998:  Include transition type 5   (kpd)

       September 2000: Add parameters T_TYPE,C_UPS,SPLUPS and comment out
               all common blocks (Vinay Kashyap)
       THIS ROUTINE IS OBSOLETE FOR CHIANTIv4+ (VK; Jun02)
```

B.38 DETECT_LIMIT [/stat]

```
function        detect_limit
        compute and return the counts upper limit for
        detection of a source at a given significance,
        given the background counts.

description
        compute the cumulative significance of obtaining a specified
        number of counts given the background, and assume that a source
        would be considered detected if the counts were to exceed the
        NSIG threshold.

        see Pease, Drake, & Kashyap (2006, ApJ 636, 426) for full description.
        briefly, computes the probability that as many as D counts can be
        observed for a given background b, p(=<D|b), and thence the probability
        that a given number of counts can be obtained in an observation simply
        due to the background.  The number of counts required for a detection
        at a specified probability is the upper limit.  Note also that this
        goes only halfway towards a full upper limit (see Kashyap et al., 2008,
        AAS-HEAD, 2008.03.03) in that this produces an upper limit in counts
        space and not in intrinsic flux space.

syntax
        ul=detect_limit(bkg,nsig,asrc=asrc,abkg=abkg,bgalt=bgalt,abgalt=abgalt,$
        nsim=nsim,ulsig=ulsig,ulsim=ulsim,/gaussy,nxbin=nxbin,verbose=verbose)

parameters
        bkg     [INPUT; required] counts in the background
        nsig    [INPUT] the Gaussian-equivalent sigma at which
                to compute upper limit to detection
                * if not given, assumed to be 3

keywords
        asrc    [INPUT; default=1] area in which source counts are collected
        abkg    [INPUT; default=1] area in which background counts are collected
        bgalt   [INPUT; default=0] alternative set of background
                contamination, say from a different area of the instrument,
                or from a model, or from an extended source, etc.
                * may be an array
                * NOTE: the _same_ BGALT is appended to _all_ elements of BKG
        abgalt  [INPUT; default=asrc] area in which BGALT is "collected"
                * size must match BGALT
                * if size < size(BGALT), first element is assumed to be
                  the default
```

```
        nsim    [INPUT; default=0] number of Monte Carlo simulations to run
                to account for error in background
                * setting this results in computing the upper limit for
                  a number of realizations of BKG; the resulting 1-sigma
                  range in the value of computed upper limits is reported
                  in ULSIG, and a conservative upper limit based on
                  combining all of the simulations is returned in ULSIM[*,0]
        ulsig   [OUTPUT] the 1-sigma error on the upper limit, estimated
                by bootstrapping BKG
        ulsim   [OUTPUT] a 2D array of size (NBKG,NSIM+1) which contains
                all the simulated limits
                * ULSIM[*,0] is identical to the primary output if NSIM=0
                * for NSIM>0, ULSIM[*,0] is the conservative limit that
                  is derived from the coadded probability distributions
                  that take into account the variations in the background
        gaussy  [INPUT] if set, computes the limit corresponding to the
                significance matching the location of the NSIG-sigma
                *intercept* of a Gaussian, rather than matching the total
                area under the curve.
        nxbin   [INPUT] number of bins to use in the integration
                * the integration is carrid out over a range of
                  0..5*E(bg) or 20, whichever is greater.
                  by default, the number of bins is set by the step size,
                  which is set to 1 count,
                  -- unless E(bg) < 1, in which case a bin width of 0.05
                  is used by default
                * changing NXBIN does not change the range, but only
                  changes the bin size.
                * a hard lower limit of 20 is set -- cannot use a bin
                  width larger than 1 count
        verbose [INPUT] controls chatter

subroutines
        LNPOISSON()
        KILROY

history
        vinay kashyap (Apr2004)
        added keyword ULSIG; changed name from PUPLIM to DETECT_LIMIT
          (VK; May2004)
        added keyword NXBIN (VK; Sep2004)
        added keywords BGALT and ABGALT (VK; Dec2004)
        modified output behavior of ULSIM[*,0]; now BGALT can be 0
          (VK; Mar2005)
        fixed bug for when NXBIN is set; made the numerical precision
```

```
problem at high NSIG explicit by making the code return -1 as
the UL (VK; Mar2009)
```

B.39 DID2EMIS2EM [/]

```
procedure       did2emis2em
        for a specified line, read in emissivities at various densities,
        compute fluxes, and invert to get emission measure estimates at
        each density.

syntax
        did2emis2em,idstr,edens,fluxes=fluxes,/deblend,DEM=DEM,logT=logT,$
        ldir=ldir,NH=NH,EMs=EMs,FXs=FXs,xtitle=xtit,ytitle=ytit,title=tit,$
        verbose=v,dWVL=dWVL,eps=eps,/incieq,mapping=mapping,chifil=chifil,$
        chidir=chidir,eqfile=eqfile,abund=abund,/noph,effar=effar,wvlar=wvlar,$
        /kev,tol=tol,fH2=fH2,He1=He1,HeII=HeII,/Fano,/wam,/bam,/mam,/Zbeda,$
        /Ibeda,Wbeda=Wbeda,/Lbeku,wform=wform,wstyle=wstyle,ziform=ziform,$
        /nuthin,/noday,/notime,/nouser,/nopack,stacol=stacol,stasiz=stasiz,$
        stathk=stathk,charsize=charsize,align=align

parameters
        idstr   [INPUT; required] structure containing ID information
                (see LINEID for description)
        edens   [I/O] array of electron densities [cm^-3]
                * default is [1e8,1e14]
                * if not specified, insufficiently specified, or otherwise
                  meaningless, uses default and overwrites input

keywords
        fluxes  [I/O] observed fluxes
                * if size does not match the number of components
                  or the number of IDs, will instead use IDSTR.(#).FLUX
                  and overwrite the input
                * see UPDATID() for details
        deblend [INPUT] if _not_ set (which is the default), squishes multiple
                IDs of a single feature into one single flux and calculates
                only a "composite" ID.  on the other hand, if set, leaves
                the multiple IDs alone.
        DEM     [INPUT] differential emission measure to use in computing
                the fluxes
                * default is to use 1e12 [cm^-5] at each specified LOGT
        logT    [INPUT] temperatures at which DEM is defined
                * if not specified, set to 6.5, unless DEM has more elements,
                  in which case is interpolated into the 4..8 range
        ldir    [INPUT] string array of database directories to use to
                search for the emissivities
                * default is '$CHIANTI'
                * if size does not match either number of IDSTR components
```

```
                  or IDs, only the first element is used
        NH      [INPUT] H column density [cm^-2]
        EMs     [OUTPUT] array of emission measures EMS[EDEN,WVL] that
                are plotted.
                * at any given point, EMS = \int DEM[LOGT]*dLOGT,
                  unless DEM has one or two elements only, in which
                  case EMS = SUM(DEM)
                * what this implies is that the input DEM is scaled
                  at each density to match the observed flux at each
                  wavelength and this scaled DEM is represented in the
                  plot by the summed measure.  This ensures that all
                  wavelengths considered are compared over the same
                  temperature range, and the ambiguity regarding the
                  temperature of peak emissivity, etc. has been removed.
        FXs     [OUTPUT] the predicted fluxes FXS[EDEN,WVL] which are used
                to scale the EMs above.
        xtitle  [INPUT] passed to PLOT
        ytitle  [INPUT] passed to PLOT
        title   [INPUT] passed to PLOT
        verbose [INPUT] controls chatter
        _extra  [INPUT] pass defined keyword values to subroutines:
                ID2EMIS2ID: DWVL
                RD_LIST: EPS, INCIEQ, MAPPING
                FOLD_IONEQ: CHIFIL, EQFILE
                RD_IONEQ: CHIDIR
                SQUISHEM: ABUND
                LINEFLX: ABUND, NOPH, EFFAR, WVLAR, KEV
                ARRAYEQ: TOL
                ISMTAU: fH2, He1, HeII, FANO, WAM, BAM, MAM
                IDLABEL: ZBEDA, IBEDA, WBEDA, LBEKU, WFORM, WSTYLE, ZIFORM
                STAMPLE: NUTHIN,NODAY,NOTIME,NOUSER,NOPACK,STACOL,STASIZ,STATHK
                XYOUTS: CHARSIZE,ALIGN
        n_e     [IGNORE] here only to trap boxing gloves

restrictions
        requires IDL5.3+ because of use of STRMATCH, etc.
        requires subroutines
          BAMABS
          CAT_LN
          FOLD_IONEQ
          GETABUND
          ID2EMIS2ID
          IDLABEL
          INICON
          ISMTAU
```

```
          LAT2ARAB
          LINEFLX
          RDABUND
          RD_IONEQ
          RD_LINE
          RD_LIST
          READ_IONEQ
          SQUISHEM
          SYMB2ZION
          SYZE
          UPDATID
          WHEE
          ZION2SYMB

side-effects
        generates a plot which erases any plot that exists beforehand

history
        vinay kashyap (JanMMI; based on DID2EM)
        improved color-scale setting for 24-bit consoles (VK; FebMMI)
```

B.40 DISCRIMINATEMP [/]

```
function        discriminatemp
        an implementation of Mark Weber's scheme for determining
        the temperature discrimination ability of a given set of
        emissivity functions.  the output is a square matrix
        of the same size as the temperature grid, showing the value
        of a statistic that can be used to infer whether two
        corresponding temperatures can be distinguished.

        the technique is simple -- compute fluxes at two temperatures
        for the same emission measure, and for an assumed uncertainty,
        ask whether the two sets of fluxes differ in a statistically
        significant manner.  if they do, the two temperatures are
        distinguishable, and not if they do not.

syntax
        dtemp=disriminatemp(emis,ferr,EM0=EM0,outflx=outflx,$
        wvl=wvl,Z=Z,logT=logT,abund=abund,normark=normark,$
        outeflx=outeflx,verbose=verbose,$
        /noph,effar=effar,wvlar=wvlar,/ikev,metals=metals,fipbias=fipbias)

parameters
        emis    [INPUT; required] array of emissivities
                * EMIS is a 2D array with size (LOGT, WVL)
                * the units are assumed to be [1e-23 ergs cm^3/s]
                  for anything else (e.g., for solar work), put the
                  difference into EM0 and be sure to set the keyword
                  NOPH=1, because it gets pushed into LINEFLX()
        ferr    [INPUT] the assumed uncertainties on the lines
                * if not given, taken to be 0.1 of the computed
                  fluxes for all lines
                * if scalar, then
                    if >0 and <1, taken to be that fraction of the
                    computed fluxes for all lines
                    if >1 and <100, taken to be that percentage of
                    the computed fluxes for all lines
                    if >100, the reciprocal is taken to be the
                    fraction of the computed fluxes for all lines
                    if <0, the abs value is taken to be a constant
                    absolute uncertainty for all lines
                * if vector, each individual value is used as given,
                  with the same condition on each as for the scalar
                  case
                * e.g., if there are 4 lines, and FERR=[-10,0.3,5,0],
```

```
                  after the 4 fluxes are computed, the corresponding
                  errors are set to [10.,0.3*F[1],0.05*F[2],0.1*F[3]]
                * the best way to set reasonable errors is to run
                  this program once, extract the computed fluxes
                  using keyword OUTFLX, figure out the appropriate
                  errors, and then feed it back into another run

keywords
        EM0     [INPUT] the emission measure to use
                * default is 1d14 cm^-5, fwiw
        outflx  [OUTPUT] the fluxes calculated for each line for each
                temperature, is an array of size (LOGT,WVL), same as EMIS
        outeflx [OUTPUT] the flux errors calculated for each line for each
                temperature, is an array of size (LOGT,WVL), same as EMIS
        wvl     [INPUT] line wavelengths for which EMIS is given
                * used only if keyword NOPH is _not_ set (i.e.,
                  in the conversion of [erg/...] to [ph/...])
                  and if size matches the 2nd dimension of EMIS
        Z       [INPUT] atomic numbers of elements contributing
                to EMIS
                * used only if size matches 2nd dimension of EMIS,
                  and is used to multiply EMIS with the appropriate
                  abundance
                * if not given, assumed to be 1 (i.e., H), which
                  is essentially a way to ignore ABUND
        logT    [INPUT] log_10(Temperature [K]) at which EMIS are given
                * unused
        abund   [I/O] element abundances
                * if size smaller than 30, calls GETABUND() and resets
                  to use Anders & Grevesse
        normark [INPUT] a normalization factor, usually set to the
                maximum number of filters or lines that can be used,
                and whose squared reciprocal divided into the output --
                Mark Weber uses this to "normalize" the results across
                different filter choices
                * hard minimum value is 1, also the default
                * if this is set, the diagonal elements of the output,
                  which are identically zero, are reset to the mean of
                  the nearest neighbors
                * if vector, only the first element is used and all the
                  extra elements are ignored
        verbose [INPUT] controls chatter

        _extra  [INPUT ONLY] pass defined variables to subroutines
                LINEFLX : /NOPH, /IKEV, EFFAR, WVLAR
```

```
                GETABUND : METALS, FIPBIAS

description
        Weber et al., 2008, SPD?

history
        vinay kashyap (2009mar)
        bigfix: wasn't dealing with vector FERR gracefully (VK; 2010aug)
```

B.41 DORREN [/timing]

```
function  dorren

 computes spot modulated stellar light curve using Dorren(1987)

syntax
      curve =  x, A, f, pder,ldstr=ldstr,ldspt=ldspt,rdns=rdns,$
      photT=photT,spotT=spotT,wvl=wvl

parameters
       x   [INPUT;required] absisscа points at which to compute spot
           model. These are angular displacements with respect to the
           initial spot longitude specified in A, in units of degrees.
       A   [INPUT;required] spot parameters for dorrens model in
                            radians where:
                            a(0) = stellar inclination to LOS
                            a(1) = spot latitude
                            a(2) = spot size
                            a(3) = initial spot longitude
       pder[OUTPUT]         partial derivative with respect to each
                            parameter in A
keywords
       ldstr [INPUT] limb-darkening coefficient for star
                          default = 0.32 Van Hamme(1994)
       ldspt [INPUT] limb-darkening coefficient for spot (umbra)
       rdns  [INPUT] set this parameter if spot parameters and absissca are input
                     in radians rather than degrees
       photT [INPUT] photospheric temperature in degrees Kelvin
       spotT [INPUT] spot temperature in degrees Kelvin
       wvl   [INPUT] wavelength in nanometers
subroutines
       MOD

history
       liwei lin (Sep 03) better documentation/functionality than
                          mod.pro. switch to modular
                 (Jul 06) removed loop over longitudes and add pder
                 (Aug 06) BUGFIX: note that T should be defined as:
                           Pi - ATAN((-SIN(h(tmp)))*(TAN(b1(tmp)))) when B1>=Pi/2
                           ATAN((SIN(h(tmp)))*(TAN(b1(tmp)))) when B1<Pi/2
                          to avoid dicontinuities. This differs from Dorren (1987)
                          formalism only in the '=' sign placement
```

B.42 DOVE_CICLO [/ssw]

```
function        dove_ciclo
        this function acts as a wrapper to all the morphological
        operations that are used to extract pixels that define a loop
        from an image of the solar corona

        the general workflow is:
        - [CONVOL] - enhance contrast by subtracting background
        - [MORFO_ROTTANGOLI] open with rectangular structure element
        - [MORPH_CLOSE] close with small square kernel
        - [CONVOL] enhance contrast again by subtracting background
        - [MORFO_SOGLIA] threshold to selectively enhance structures
        - [MORPH_CLOSE] close the image again
        - [MORFO_SEGMENTO] group contiguous pixels into region blobs
          and throw out regions deemed too small
        - [ROI_SELEZIONI] interactively select some blobs for further study
        - [MORFO_SCHELETRO] create a skeleton for these blobs
        - [MORFO_POTARE] prune the skeleton to make a cleaner image
        - [] extract the pixels that form the pruned skeleton and
          convert them to heliographic coordinates

        keywords control whether any of the calls to the subroutines
        should be skipped or repeated.  keywords also set up the input
        parameters to the subroutines.

syntax
        oimg=dove_ciclo(inpimg,$
        /xbgdat,icell=icell,$
        /xrect,xsize=xsize,ysize=ysize,angle=angle,gray=gray,/readd,/reuse,$
        /xrclose,$
        /xbgsub,jcell=jcell,$
        /xthresh,thrthr=thrthr,nsigma=nsigma,mimg=mimg,gamma=gamma,$
                centrale=centrale,bitlev=bitlev,/zeromin,$
        /xtclose,$
        /xblob,blbthr=blbthr,areas=areas,$
                subidx=subidx,/hardcut,deepimg=deepimg,$
        /xroi,pixroi=pixroi,roithr=roithr,jitter=jitter,/hadrian,$
        /xskel,skthr=skthr,rmin=rmin,$
        /xprune,minpix=minpix,$
        outx=outx,outy=outy,sunrad=sunrad,pixsiz=pixsiz,$
        offsetx=offsetx,offsety=offsety,suncenx=suncenx,sunceny=sunceny,$
        loopsav=loopsav, verbose=verbose)

parameters
```

```
        inpimg  [INPUT; required] image of the corona

keywords        For keywords with names beginning with "X", when set,
                the corresponding call to the subroutine is skipped.
                But if that keyword is set to a negative number -N,
                the subroutine is called N times in sequence (unless
                otherwise noted below).

        xbgdat  [INPUT] if set, skips initial background subtraction
        icell   [INPUT; default=1] defines the size of the cell to subtract
                background - the "source" is measured in a square of size
                2*ICELL+1 and the "background" in a surrounding layer of 1pix
        xrect   [INPUT] if set, skips morphological open with rectangular
                structure element
        xsize   [INPUT; default=1] width of the structure element
        ysize   [INPUT; default=10] height of the structure element
        angle   [INPUT; default=findgen(36)*5] angle of tilt of the rectangle
                defined by (XSIZE,YSIZE) in [degrees]
        xrclose [INPUT] if set, skips morphological close apres open
        xbgsub  [INPUT] if set, skips background subtraction of modified image
        jcell   [INPUT; default=ICELL] defines size of cell to subtract
                background
        xthresh [INPUT] if set, skips image thresholding
        thrthr  [OUTPUT] value used if histogram-threshold is applied
        nsigma  [INPUT] if given, sets the threshold at mean+NSIGMA*stddev
                * default is 1
                * if mean+NSIGMA*stddev < 0, this is not applied
        mimg    [OUTPUT] the median threshold image, constructed if
                CENTRALE is set
        xtclose [INPUT] if set, skips morphological close apres thresholding
        xblob   [INPUT] if set, skips grouping the image into distinct regions
                * if set to -ve number, gets automatically unset
        areas   [OUTPUT] number of pixels in each labeled region
        arclev  [INPUT; default=0.68] fraction of the area of the largest
                blob as the threshold below which to discard blobs
        deepimg [OUTPUT] an image that depicts the depth at which a
                particular pixel was added to a blob
        blbthr  [INPUT; default=0] threshold value at which to convert
                grouped image to bitmap
        xroi    [INPUT] if set, skips selecting a subset of regions
                * if set to -ve number, gets automatically unset
        pixroi  [INPUT] if set, assumed to be a set of pixels around
                which to pick out the region of interest non-interactively
        roithr  [INPUT; default=0] threshold value at which to ignore pixels
        xskel   [INPUT] if set, skips making skeletons of regions
```

```
                * if set to -ve number, gets automatically unset
        rmin    [INPUT; default=1] radius of circle that acts as
                structure element
        skthr   [INPUT; default=0] threshold to filter region
        xprune  [INPUT] if set, skips pruning skeletons
                * if set to -ve number, gets automatically unset
        minpix  [INPUT; default=4] number of pixels above which a given
                branch must be kept while pruning
        outx    [OUTPUT] the x-pixel indices of the pruned points
        outy    [OUTPUT] the y-pixel indices of the pruned points
                * if SUNRAD is set, then floating point values defining
                  the locations in heliospheric coordinates are returned
        sunrad  [INPUT] radius of the Sun as seen from the spacecraft,
                in [arcsec]
                * if this is set, then OUTX and OUTY are transformed
                  to heliospheric coordinate system
                * if set to 1, assumed to be (RSun/AU)*(180/!pi)*3600
        pixsiz  [INPUT; default=0.5] pixel size, in [arcsec]
        offsetx [INPUT; default=0] offset to be applied to OUTX
                to bring them in line with SUNCENX
        offsety [INPUT; default=0] offset to be applied to OUTY
                to bring them in line with SUNCENY
        suncenx [INPUT; default=0] the X coordinate distance of the
                center of INPIMG from Sun center in [arcsec]
        sunceny [INPUT; default=0] the Y coordinate distance of the
                center of INPIMG from Sun center in [arcsec]
        verbose [INPUT] controls chatter
        loopsav [INPUT] if set to a filename, saves all the
                intermediate arrays in the named IDL save file
        _extra  [INPUT ONLY] pass defined keywords to subroutines
                MORFO_ROTTANGOLI: GRAY, READD, REUSE
                MORFO_SOGLIA: GAMMA, CENTRALE, BITLEV, ZEROMIN
                MORFO_SEGMENTO: SUBIDX, HARDCUT
                ROI_SELEZIONI: JITTER, HADRIAN

subroutines
        MORFO_ROTTANGOLI
        MORFO_SOGLIA [GMASCL,ERROR_MESSAGE,PATH_SEP,SCALE_VECTOR,CONVERT_TO_TYPE,FPUFIX]
        MORFO_SEGMENTO
        ROI_SELEZIONI
        MORFO_SCHELETRO
        MORFO_POTARE
        INICON
        KILROY
```

```
history
        vinay kashyap (May2007)
```

B.43 DRAKOPY [/util]

```
procedure       drakopy
        set up the environment to make plots per individual preferences

syntax
        drakopy,milieu,/HELP,/UNDO,MAGNIFY=MAGNIFY,VERBOSE=VERBOSE

parameters
        milieu  [INPUT] string specifying what sort of environment to set to.
                accepted values are:
                -- 'jeremy'
                -- 'olivia'
                -- 'jjd' (jeremy old preferences)
                -- 'drake' (jeremy older preferences)
                -- 'vinay' (different depending device at call time)
                -- 'poster'
                -- 'original'

keywords
        help    [INPUT] if set, prints out a somewhat verbose help
        undo    [INPUT] if set, restores original state
        verbose [INPUT] controls chatter level
        magnify [INPUT] scale all the variables by this factor
        _extra  [JUNK] here only to prevent crashing the program

commons
        drakopy {old_P, old_X, old_Y, old_Z}

description
        sets the !P, !X, !Y, and !Z variables to special and specific values.
        stores the old values in a common block for later access.

example
        set_plot,'ps' & device,file='/tmp/test_drakopy.ps'
          plot,findgen(10),title='MILIEU=Original'
          drakopy,'jeremy' & plot,findgen(10),title='MILIEU=Jeremy'
          drakopy,'olivia' & plot,findgen(10),title='MILIEU=Olivia'
          drakopy,'original' & plot,findgen(10),title='MILIEU=Original'
          drakopy,/undo & plot,findgen(10),title='MILIEU=previous (Olivia)'
        device,/close & set_plot,'x'

history
        vinay kashyap (SepMM)
        added option "drake" (VK; Aug01)
```

```
added option "jjd"; added keyword MAGNIFY (VK; Feb02)
added option "vinay" (VK; May03)
added option "poster" (VK; Jul03)
told the code that "poster" is a valid option (VK; Sep03)
now prints out help and exits if no input is given (VK; Jul13)
changed "jeremy" defaults (JD; Nov15)
```

B.44 DRAT [/]

```
function        drat
        return the ratio of fluxes in one line relative to another line
        for a variety of densities

syntax
        frat=drat(line1,line2,edens,DEM,logT,fx1=fx1,fx2=fx2,$
        mapping=mapping,chifil=chifil,chidir=chidir,eqfile=eqfile,$
        abund=abund,/noph,effar=effar,wvlar=wvlar,NH=NH,fH2=fH2,$
        He1=He1,HeII=HeII,/Fano,dbdir=dbdir,sep=sep,prefix=prefix)

parameters
        line1   [INPUT; required] string describing the line/wavelength-range
                that defines the numerator
        line2   [INPUT; required] string describing the line/wavelength-range
                that defines the denominator
                * LINE1 and LINE2 must be in the format used by RD_LIST, i.e.,
                        Z ION <sep> WAVE <sep> DBDIR <sep> DESCRIPTION
                  and WAVE = "WVL"/"WVL +- dW"/"WMIN,WMAX"/"WMIN-WMAX"
        edens   [INPUT; required] electron densities at which to compute
                the flux ratios
        DEM     [INPUT; required] differential emission measure to use in
                computing the fluxes
        logT    [INPUT; required] temperatures at which DEM is defined
                * size must match that of DEM

keywords
        fx1     [OUTPUT] the fluxes due to line 1 at each EDENS
        fx2     [OUTPUT] the fluxes due to line 2 at each EDENS
        _extra  [INPUT ONLY] pass defined keywords to subroutines
                RD_LIST: MAPPING,DBDIR,SEP,PREFIX
                FOLD_IONEQ: CHIFIL,EQFILE
                RD_IONEQ: CHIDIR
                LINEFLX: ABUND,NOPH,EFFAR,WVLAR
                ISMTAU: NH,fH2,He1,HeII,FANO,BAM
                BAMABS: ABUND

subroutines
        RD_LIST
        RD_LINE
        FOLD_IONEQ
        LINEFLX
        ISMTAU
        BAMABS
```

```
        CAT_LN

history
        vinay kashyap (MarMM)
```

B.45 DUMMYID [/]

```
function  dummyid
        Convert a RD_LINE() style emissivity structure into a dummy
        ID structure such as that generated by LINEID()

syntax
        idstr=dummyid(linstr,DEM=DEM,/temp,abund=abund,/noph,$
        nhne=nhne,effar=effar,wvlar=wvlar,/ikeV,/noabund)

parameters
        linstr  [INPUT; required] RD_LINE() or RD_LIST() style line
                emissivity structure

keywords
        _extra  [INPUT ONLY] use this to pass defined keywords to subroutines
                LINEFLX: DEM,ABUND,EFFAR,WVLAR,NOABUND,NHNE,NOPH,TEMP,IKEV

subroutines
        LINEFLX()
history
        liwei lin (Feb06)  lift parameter check from cat_ln()
        cleaned up a bit (VK; Feb06)
```

B.46 EG_IDSTR [/esempio]

```
EG_IDSTR.PRO
        an example program that extracts information out of an ID structure

        requires an input ID structure in the variable IN_IDSTR
        extracts all the ID'd wavelengths, atomic numbers, ionic states,
        and places them in arrays named OUT_<var>
        look in OUT_IDX for a pointer to the original component

        TBD: need to be cleverer about labels and notes.

subroutines
        ZION2SYMB

vinay kashyap (AugMM)
```

B.47 EQT_INTERVAL [/stat]

```
function        eqt_interval
        computes and returns the double-sided equal-tail interval
        [lower_bound,upper_bound] at a specified confidence level.
        interval includes the central mass of the probability
        distribution unless explicitly required otherwise.

syntax
        hpd=eqt_interval(f,x,/fsample,clev=clev,pdfnorm=pdfnorm,$
        xaround=xaround,verbose=verbose)

parameters
        f       [INPUT; required] the array for which the confidence interval
                must be computed
                * assumed to be a density function unless FSAMPLE is set
        x       [INPUT; optional] abscissae values
                * if not given, and F is a density function, then taken
                  to be the array indices
                * ignored if FSAMPLE is set
                * if N(X).GT.1 but N(X).NE.N(F), X is ignored and
                  FSAMPLE is set automatically

keywords
        fsample [INPUT] if set, assumes that F is a set of samples from a
                density function, as opposed to being the density function
                itself
        clev    [INPUT] confidence level at which to compute the intervals
                * default is 0.68
                * if < 0, abs(CLEV) is used
                * if > 1 and < 100, then assumed to be given as a percentage
                * if > 100, then 1-1/CLEV is used
        pdfnorm [INPUT] if set, forces F to integrate to abs(PDFNORM)
                * if explicitly set to 0, does not normalize F at all
                * if not set, normalizes to 1
                * ignored if FSAMPLE is set
                * WARNING: do not use this keyword unless you know
                  what you are doing
        xaround [INPUT] by default, the central part of the distribution
                is used, i.e., an equal area is left out on both ends.  if
                this keyword is defined, a fraction CLEV/2 of the area is
                used on either side of XAROUND.
                * if F is a sample, XAROUND must be in the same units as F
                * you can return single-sided intervals by setting XAROUND
                  to min(X) or max(X) (min/max of F if sample)
```

```
                * if not set, XAROUND is set to the median
        verbose [INPUT] controls chatter
        _extra  [INPUT ONLY] pass defined keywords to subroutines

example
        for i=1,20 do print,eqt_interval(randomn(seed,10000L)*i,/fsample)

history
        vinay kashyap (Apr2006)
```

B.48 ERORS [/fitting]

```
procedure       erors
        compute asymmetric error bars on fit parameters by projecting the
        chi-sq surface onto each parameter axis.  what this means is that
        for each non-frozen parameter, step through a range of values of
        the parameter, finding the best-fit chi-square calculated by fitting
        the rest of the parameters, and report that range where the chi-square
        increases by a set amount from the minimum.

        this program tries to be a wee bit clever by starting from the
        best-fit value and then stumbling about on either side until
        the chi-sq goes above the mark; the length of the strides change
        according to the projected location of said mark.

syntax
        erors,x,y,a,erru,errl,ysig=ysig,freeze=freeze,dchi=dchi,$
          dchi=dchi,algo=algo,maxstep=maxstep,verbose=verbose,$
          erra=erra,yfunc=yfunc,$
          itmax=itmax,chithr=chithr,/dumb, jumpup=jumpup,jumpdn=jumpdn,$
          svdthr=svdthr,funcs=funcs, function_name=function_name,$
          type=type, missing=missing,/fwhm,/norm,betap=betap, /poisson

parameters
        x       [INPUT; required] data points
        y       [INPUT; required] Y(X)
                * sizes of X and Y must match
        a       [I/O; required] parameters for user-supplied function
                * on input, these are assumed to be initial guesses
                * on output, these contain the best-fit values
        erru    [OUTPUT; required] upper limits of confidence range interval
        errl    [OUTPUT] lower limits of confidence range interval
                * if ERRL is not specified on input, ERRU will contain the
                  average value of the upper and lower >>deviations<<.

keywords
        ysig    [INPUT] standard deviations on Y
                * default=sqrt(abs(Y)+0.75)+1
                * if single element, then sig(Y[*])=YSIG(0)
                * if -ve, taken to be the fractional error
        freeze  [INPUT] freeze numbered parameters (index starts from 0!)
        dchi    [INPUT] how big a change in chi-square to look for?
                * default is 2.7 (corresponding to 90% CL)
        algo    [INPUT] fitting algorithm
                * only the following are implemented:
```

```
                -- LevMarq+SVD (default; calls FIT_LEVMAR)
                -- IDL-Curvefit (calls CURVE_FIT)
        maxstep [INPUT] maximum number of steps to take before giving up
                on actually finding the bounds
                * default is 100
        verbose [INPUT] verbosity level
        erra    [OUTPUT] formal "curvature" errors on the best-fit parameters
        yfunc   [OUTPUT] best-fit Y(X;A)
        _extra  [INPUT] pass defined variables to subroutines:-
                FIT_LEVMAR: ITMAX, CHITHR, DUMB
                ADJUSTIE: TIES, VNAME
                LEVMARQ: JUMPUP, JUMPDN, SVDTHR
                LMCOEFF: FUNCS, POISSON
                CURVE_FIT: FUNCTION_NAME
                note:-  FUNCS and FUNCTION_NAME refer to name of
                        user-defined function that takes as input X
                        and A, and returns YMODEL(X;A), and the
                        partial derivatives of the model function
                        wrt A.  Any function that was written to work
                        with CURVEFIT or GHRS' WFIT will do.
                        The default for FIT_LEVMAR is X3MODEL.
                MK_3MODEL: TYPE
                MK_GAUSS: MISSING, FWHM, NORM
                MK_LORENTZ: BETAP, MISSING, NORM

history
        vinay kashyap (MM.I) (yes, I _do_ know how to spell "error")
        added call to ADJUSTIE to handle constraints on ERRU,ERRL (VK; FebMM)
        what if adjusted ERRA becomes 0? (VK; MarMM)
        allowed halting with either "q" or "x" (VK; SepMM)
        force extra iteration after "r"; increased wait time (VK; JanMMI)
        added confirmation check for too many thawed params (VK; FebMMI)
        interpol was crashing because it was getting only 1 element
          arrays; now correctly updates all parameters if better fit is found
          (VK; Aug01)
        made various bug fixes that was causing program to go bonkers for
          some special cases, such as small fluxes (VK; Apr02)
        added hooks into MPFIT (Liwei Lin/VK; Oct02)
        bug correction re VERBOSE (LL; Apr03)
        bug correction, first step was failing sometimes when input A were
          integers (VK; Mar08)
```

B.49 ERROR_MESSAGE [/external]

```
NAME:
   ERROR_MESSAGE

PURPOSE:

   The purpose of this function  is to have a device-independent
   error messaging function. The error message is reported
   to the user by using DIALOG_MESSAGE if widgets are
   supported and MESSAGE otherwise.

   In general, the ERROR_MESSAGE function is not called directly.
   Rather, it is used in a CATCH error handler. Errors are thrown
   to ERROR_MESSAGE with the MESSAGE command. A typical CATCH error
   handler is shown below.

      Catch, theError
      IF theError NE 0 THEN BEGIN
         Catch, /Cancel
         ok = Error_Message()
         RETURN
      ENDIF

   Error messages would get into the ERROR_MESSAGE function by
   throwing an error with the MESSAGE command, like this:

      IF test NE 1 THEN Message, 'The test failed.'

AUTHOR:

  FANNING SOFTWARE CONSULTING
  David Fanning, Ph.D.
  1645 Sheely Drive
  Fort Collins, CO 80526 USA
  Phone: 970-221-0438
  E-mail: davidf@dfanning.com
  Coyote's Guide to IDL Programming: http://www.dfanning.com/

CATEGORY:

   Utility.

CALLING SEQUENCE:
```

```
   ok = Error_Message(the_Error_Message)

INPUTS:

   the_Error_Message: This is a string argument containing the error
      message you want reported. If undefined, this variable is set
      to the string in the !Error_State.Msg system variable.

KEYWORDS:

   ERROR: Set this keyword to cause Dialog_Message to use the ERROR
      reporting dialog. Note that a bug in IDL causes the ERROR dialog
      to be used whether this keyword is set to 0 or 1!

   INFORMATIONAL: Set this keyword to cause Dialog_Message to use the
      INFORMATION dialog instead of the WARNING dialog. Note that a bug
      in IDL causes the ERROR dialog to be used if this keyword is set to 0!

   TITLE: Set this keyword to the title of the DIALOG_MESSAGE window. By
      default the keyword is set to 'System Error' unless !ERROR_STATE.NAME
      equals "IDL_M_USER_ERR", in which case it is set to "Trapped Error'.

   TRACEBACK: Setting this keyword results in an error traceback
      being printed to standard output with the PRINT command. Set to
      1 (ON) by default. Use TRACEBACK=0 to turn this functionality off.

OUTPUTS:

   Currently the only output from the function is the string "OK".

RESTRICTIONS:

   The WARNING Dialog_Message dialog is used by default.

EXAMPLE:

   To handle an undefined variable error:

   IF N_Elements(variable) EQ 0 THEN $
      ok = Error_Message('Variable is undefined')

MODIFICATION HISTORY:

   Written by: David W. Fanning, 27 April 1999.
   Added the calling routine's name in the message and NoName keyword. 31 Jan 2000. DWF.
```

```
Added _Extra keyword. 10 February 2000. DWF.
Forgot to add _Extra everywhere. Fixed for MAIN errors. 8 AUG 2000. DWF.
Adding call routine's name to Traceback Report. 8 AUG 2000. DWF.
Added ERROR, INFORMATIONAL, and TITLE keywords. 19 SEP 2002. DWF.
Removed the requirement that you use the NONAME keyword with the MESSAGE
  command when generating user-trapped errors. 19 SEP 2002. DWF.
Added distinctions between trapped errors (errors generated with the
  MESSAGE command) and IDL system errors. Note that if you call ERROR_MESSAGE
  directly, then the state of the !ERROR_STATE.NAME variable is set
  to the *last* error generated. It is better to access ERROR_MESSAGE
  indirectly in a Catch error handler from the MESSAGE command. 19 SEP 2002. DWF.
Change on 19 SEP 2002 to eliminate NONAME requirement did not apply to object method
  Fixed program to also handle messages from object methods. 30 JULY 2003. DWF.
Removed obsolete STR_SEP and replaced with STRSPLIT. 27 Oct 2004. DWF.
Made a traceback the default case without setting TRACEBACK keyword. 19 Nov 2004. DW
```

B.50 EXAPED [/mkemis]

```
procedure       exaped
        EX APED AD IDL
        extract line and/or continuum emissivity data from an APED file

syntax
        exaped,filroot,emis,tlog,wvl,edens,Z,ion,desig,econf,jon,src,$
        /llist,/coco,/tlist,/noerg,/okeV,atomdb=atomdb,verbose=verbose

parameters
        filroot [INPUT; required] APED fits file root
                * if this includes a ".fits", the suffix is assumed
                  to be already included and nothing extra is added.
                * the suffix '_linelist.fits' is automatically applied,
                  so don't specify that part.
                  (this is inflexible at this point because we only
                  support reading in this file.  sometime later on
                  the other APED format will also be supported.)
                * looks for ".gz" automatically if the filename
                  specified without it does not exist
        emis    [OUTPUT; required] emissivity array, of the format
                (NTLOG,NWVL,NEDENS)
                * APED units are [ph cm^3/s], which will be converted
                  into [1e-23 ergs cm^3/s] unless the NOERG keyword
                  is set
        tlog    [OUTPUT; required] log10(Temperature [K]) grid on
                which EMIS are placed
        wvl     [OUTPUT; required] wavelength array
                * units are in [Ang] unless the OKEV keyword is set,
                  in which case output in [keV]
        edens   [OUTPUT; required] electron densities at which
                EMIS have been calculated
        Z       [OUTPUT; required] atomic numbers of species producing
                the given line
        ion     [OUTPUT; required] ionic state of species corresponding
                to the line
        desig   [OUTPUT] (2,NWVL) string array of lower and upper level
                designation
                * currently, just the lower and upper levels
        econf   [OUTPUT] (2,NWVL) string array of electron configuration
                of lower and upper levels of the transition in question
                * not implemented yet
        jon     [OUTPUT] the ionic state that matters to the ion balance
                * not implemented yet
```

```
        src     [OUTPUT] numeric code specifiying that this is an APED
                output, set to 3

keywords
        llist   [INPUT] if set, reads from FILROOT_linelist.fits
                * currently this is the only option supported
        coco    [INPUT] if set, reads from FILROOT_coco.fits
                * NOT YET IMPLEMENTED
        tlist   [INPUT] if set, reads from FILROOT_line.fits
                * NOT YET IMPLEMENTED
        noerg   [INPUT] if set, EMIS will be output in same units as
                in the APED files, i.e., [ph cm^3/s]
        okeV    [INPUT] if set, WVL will be converted from [Ang] to [keV]
        atomdb  [INPUT] directory containing input file
                * default is !ATOMDB
                * hardcoded default is /data/atomdb/
                * prepended to FILROOT only if FILROOT does not begin with
                  a '/' (UNIX), ':' (MacOS), '\' (Windows), or '[' (VMS?)
        logT    [INPUT] if defined as an array, then interpolates/extrapolates
                EMIS(NTLOG,NWVL) to a new grid EMIS(NLOGT,NWVL) prior to
                output.  TLOG will be overwritten by LOGT.
        verbose [INPUT] controls chatter
        _extra  [JUNK] ignore -- here only to prevent crashing the program

subroutines
        SETSYSVAL
        ARRAYEQ
        REBINX

description
        The ATOMDB file format is described in
                http://cxc.harvard.edu/atomdb/features_formats.html
        In this case, the line list (grouped by line) file is read, and
        in particular, only the block EMISSIVITY is read.

history
        vinay kashyap (Jul02)
        bug correction when multiple densities are included (VK; Mar03)
        for v2+, always go through the multiple densities code and ignore
          the short cut (VK; May11)
```

B.51 EXIST [/external]

```
Project     :  SDAC

Name        :  EXIST

Purpose     :  To See if variable Exists

Explanation :  So obvious, that explaining it will take more
              lines than the code.

Use         :  A=EXIST(VAR)

Inputs      :  VAR = variable name

Opt. Inputs : None.

Outputs     :  1/0 if variable exists or not

Opt. Outputs:  None.

Keywords    :  None.

Calls       :  None.

Common      :  None.

Restrictions:  None.

Side effects:  None.

Category    :  Useful stuff

Prev. Hist. :  None.

Written     :  Dominic Zarro (ARC)

Version     :  Version 1.0, 18 September 1993
```

B.52 FIDGIT [/fitting]

```
procedure       fidgit
        digitally (i.e., using Darwin-given fingers) fit a DEM
        to a given spectrum

syntax
        fidgit,x,y,lstr,cstr,ysigma=ysigma,dem=dem,logt=logt,abund=abund,$
        abfrac=abfrac,ldbdir=ldbdir,cdbdir=cdbdir,ceroot=ceroot,lsf=lsf,$
        wdem=wdem,wspc=wspc,verbose=verbose,$
        pres=pres,logP=logP,n_e=n_e,desig=desig,econf=econf,/allah,$
        chidir=chidir,chifil=chifil,chidir=chidir,eqfile=eqfile,$
        effar=effar,wvlar=wvlar

parameters
        x       [INPUT; required] wavelengths or channels of observed
                spectrum
                * if channels, must specify an RMF via the keyword LSF
                  that matches the binning of the spectrum
                  BEWARE: no checks are performed to verify that the
                  supplied RMF is consistent with the input data
        y       [INPUT; required] spectrum
        lstr    [I/O; required] line emissivity structure of the sort read in
                by RD_LINE()
                * if not given on input, will call RD_LINE() and generate it
        cstr    [I/O; required] continuum emissivity structure of the sort
                read in by RD_CONT()
                * if not given on input, will call RD_LINE() and generate it

keywords
        ysigma  [INPUT] errors on Y; default is 1+SQRT(0.75+ABS(Y))
        dem     [I/O] DEM to start out with, will contain the final
                result on output
        logt    [I/O] log(T [K]) at which DEM are defined.  ignored on
                input if it doesn't match the size of DEM
        abund   [INPUT] abundances
                * if not set, assumed to be from Anders & Grevesse 1989
        abfrac  [I/O] multiplicative factor by which to modify ABUND
                * if scalar, assumed to be metallicity
                * if vector, must match size of ABUND, else gets reset to 1
                * default is 1
        ldbdir  [INPUT] path to line emissivity database to read in if LSTR
                is not given
                * default is '$CHIANTI'
                * uses !LDBDIR if set
```

```
        cdbdir  [INPUT] path to continuum emissivity database to read in if
                CSTR is not given
                * default is '$CONT'
                * uses !CDBDIR if set
        ceroot  [INPUT] prefix for continuum emissivity files
                * default is 'cie'
                * uses !CEROOT if set
        lsf     [INPUT] line spread function
                * if scalar, the width of the line in bins used as a
                  halfwidth for boxcar smoothing
                * if vector, assumed to be the kernel with which to smooth
                * if RMF structure (of the type read in from RD_OGIP_RMF())
                  then convolves the theoretical spectrum with this RMF
        wdem    [INPUT] window number to display DEM in
        wspc    [INPUT] window number to display spectrum in
        verbose [INPUT] controls chatter
                * uses !VERBOSE if set
        _extra  [INPUT] allows specifying defined keywords to subroutines
                called by this program
                * RD_LINE: PRES, LOGP, N_E, DESIG, ECONF, ALLAH
                * RD_CONT: PRES, LOGP, N_E
                * FOLD_IONEQ: CHIFIL, VERBOSE
                * RD_IONEQ: CHIDIR, EQFILE
                * LINEFLX: EFFAR, WVLAR

restrictions
        * requires subroutines
          -- DEMACS
          -- WHEE
          -- RD_LINE [FOLD_IONEQ, RD_IONEQ, READ_IONEQ]
          -- RD_CONT
          -- LINEFLX
          -- CONV_RMF
          -- INICON

history
        vinay kashyap (Feb97)
        modified ion balance calcs (VK; 99May)
        completely rewritten (VK; Aug04)
```

B.53 FILE_EXIST [/external]

```
Name: file_exist

Purpose: returns true(1) if any files match pattern=file
          false(0) if no files found

Input Parameters:
   file - file, pathname or file filter to check

Optional Keyword Parameters
   direct - set if want full check (slower) - use if might be an
             empty directory you are looking for

History: written by slf, 11/30/91
     4-Sep-92 (MDM) - Modified to handle an array of input files
   23-sep-93 (slf) - make sure null file returns false
                      (findfile count=1! when null requested)
   17-Mar-94 (SLF) - use file_stat(/exist) for non wild card cases
   April Fools'94 (DMZ) -- added check for :: in file name
                           (file_stat doesn't work across DECNET)
   31-May-94 (DMZ)      -- added check for VMS directory name input
                           (file_stat doesn't seem to work on VMS
                            dir names like '[ys.atest.soft]')
   16-Aug-94 (DMZ)      -- added another VMS patch to handle
                           case when user enters  a VMS subdirectory name
                           without a file name.
   6-Aug-97 (Zarro/GSFC) -- added check for nonstring input
   30-may-99 (rdb)      -- stop use of file_stat with WINDOWS
                           cause access violation - for wildcard search
   8-Jan-1999 - S.L.Freeland - put in IDL version check since 5.3/IRIX
              (at least) not backwardly compatible (must use findfile)
  10-Jan-1999 - S.L.Freeland - extended 8-jan mod to all UNIX 5.3
  15-Feb-2000 - S.L.Freeland - removed 5.3 changes (moved fix to file_stat)
  10-Feb-2005 - Kim Tolbert - changed '/' to get_delim()
```

B.54 FILE_STAT [/external]

```
Project     :  SOHO - SSW

Name        :  FILE_STAT()

Purpose     :  Vector version of FSTAT

Category    :  Utility, Operating_system

Explanation :  Vector version of FSTAT

Syntax      :  Result = FILE_STAT( FILES )

Examples    :

Inputs      :  FILES   = List of files to check.

Opt. Inputs :  None.

Outputs     :  None.

Opt. Outputs:  None.

Keywords    :  EXIST   = If set, then the returned values are whether the
                          files exist or not.  This is the default behavior.
               SIZE    = If set, then the returned values are the sizes of the
                          files.
              FSTAT_ALL = If set, return vector of 'fstat' structures.  This
                         keyword is only compatible with files less than about
                         2 GB each in size.

Calls       :  DATA_CHK

Common      :  None.

Restrictions:  None.

Side effects:  None.

Prev. Hist. :  11-Mar-1994 (SLF) Wanted faster file_exist function

History     :  Version 1, 11-Mar-1994, S. Freeland
              Version 1.1 05-Jun-1998, J. Newmark changed loop to long
              Version 1.2 15-Feb-2000, S.L.Freeland - work around RSI
```

```
                Version 5.3 non-backwardly compatible change....
              Version 1.3 10-Mar-2000, S.L.Freeland - fix 5.3 /SIZE bug
              Version 1.4 14-Jun-2002, S.L.Freeland - add FSTAT_ALL
              Version 2, 1-Mar-2005, W.T. Thompson - Add support for files
                                                    greater than 2 GB

Contact     :  SFREELAND
Restrictions:
  file size returned for directories under +5.3 is not valid
```

B.55 FIND_WVL_NOLINE [/esempio]

```
find_wvl_noline.pro
        find wavelength bins which are line-free in a given spectrum

        reads in the line emissivity database, makes a frequency histogram
        of the lines in each wavelength bin, and identifies bins where the
        number of lines is 0.

requires
        WGRID == a wavelength grid
                (if not given, a standard MEG grid of 1:41.96:0.005 is assumed)
        LDBDIR == location of line emissivity database
                (if not given, assumed to be $APED)
        LSFWDT == the expected spread in the line.  This is basically the
                window half-width of a boxcar smoothing function.
                If integer, assumed to be in pixels, and if float, taken to be
                in units of bin width and converted to pixels accordingly.
                (if not given, assumed to be 2)

vinay k (2011may26)
```

B.56 FINDEX [/external]

```
 ROUTINE:  findex

 PURPOSE:  Compute "floating point index" into a table using binary
           search.  The resulting output may be used with INTERPOLATE.

 USEAGE:   result = findex(u,v)

 INPUT:
   u       a monitically increasing or decreasing 1-D grid
   v       a scalor, or array of values

 OUTPUT:
   result  Floating point index. Integer part of RESULT(i) gives
           the index into to U such that V(i) is between
           U(RESULT(i)) and U(RESULT(i)+1).  The fractional part
           is the weighting factor

                          V(i)-U(RESULT(i))
                          ---------------------
                     U(RESULT(i)+1)-U(RESULT(i))


 DISCUSSION:
           This routine is used to expedite one dimensional
           interpolation on irregular 1-d grids.  Using this routine
           with INTERPOLATE is much faster then IDL's INTERPOL
           procedure because it uses a binary instead of linear
           search algorithm.  The speedup is even more dramatic when
           the same independent variable (V) and grid (U) are used
           for several dependent variable interpolations.


 EXAMPLE:

; In this example I found the FINDEX + INTERPOLATE combination
; to be about 60 times faster then INTERPOL.

  u=randomu(iseed,200000) & u=u(sort(u))
  v=randomu(iseed,10)     & v=v(sort(v))
  y=randomu(iseed,200000) & y=y(sort(y))

  t=systime(1) & y1=interpolate(y,findex(u,v)) & print,systime(1)-t
  t=systime(1) & y2=interpol(y,u,v)            & print,systime(1)-t
```

```
 print,f='(3(a,10f7.4/))','findex:   ',y1,'interpol: ',y2,'diff:     ',y1-y2

AUTHOR:   Paul Ricchiazzi                        21 Feb 97
          Institute for Computational Earth System Science
          University of California, Santa Barbara
          paul@icess.ucsb.edu

REVISIONS:
```

B.57 FINDSCALE [/]

```
function        findscale
        returns the lengthscale in pixels at each point on the given curve

syntax
        ls=findscale(curve,dim,/crunch,/half,pick=pick,choice=choice,eps=eps)

parameters
        curve   [INPUT; required] regularly gridded array of function
                values to be used to compute the length scales
                * if scalar, returns 0
                * if 2D,
                  -- use DIM to specify primary dimension
                  -- compute lengthscales separately along each projection
                * if >2D, convert to 1D
        dim     [INPUT; default=1] primary dimension in case of 2D array
                (e.g., if CURVE=CURVE(NX,NY), DIM=2 returns SCALE=SCALE(NY))

keywords
        crunch  [INPUT] if set, and CURVE is 2D, collapses the array along
                the secondary dimension to generate 1D curve
        half    [INPUT] if set, returns the half-scale
        pick    [INPUT; default=0] if 2D, specifies how to combine the
                scales computed at the different cuts
                0: pick the smallest scale
                1: pick the largest scale
                2: get the average
        choice  [INPUT; default=0] what algorithm to use to find the scale?
                0: MexicanHat wavelet
                1: use inverse of 1st derivative
                2: radius of curvature
                3: stepped toggle
        eps     [INPUT; default=1e-7] small number
        _extra  [JUNK] ignore.  here only to prevent crashing program.

subroutines
        WVLT_SCALE [ROOFN]

history
        vinay kashyap (Apr97)
        added CHOICE option 3 (VK; Feb03)
```

B.58 FIT_LEVMAR [/fitting]

```
procedure       fit_levmar
        uses the Levenberg-Marquardt method to find best-fit parameters

syntax
        fit_levmar,x,y,a,yfunc,freeze=freeze,erra=erra,chisq=chisq,$
        itmax=itmax,chithr=chithr,/dumb,ties=ties,vname=vname,$
        jumpup=jumpup,jumpdn=jumpdn,svdthr=svdthr,funcs=funcs,sig=sig,$
        /poisson

parameters
        x       [INPUT; required] data points
        y       [INPUT; required] Y(X)
                * sizes of X and Y must match
        a       [I/O; required] parameters for user-supplied function
                * on input, these are the initial guesses
                * on output, these contain the best-fit values
        yfunc   [OUTPUT] best-fit Y(X;A)

keywords
        freeze  [INPUT] freeze numbered parameters (starting from 0!)
        erra    [OUTPUT] formal errors on the best-fit parameters
        chisq   [OUTPUT] the chi-sq statistic denoting degree of agreement
                between model and data
        itmax   [INPUT; default=100] maximum number of iterations
        chithr  [INPUT; default=0.1] stopping rule: ignore changes in CHISQ
                smaller than this amount
        dumb    [INPUT] if set, skips the part where the user can readjust
                the parameter values in case the fit has gone bad.
        tiptoe  [INPUT] if set, forces small steps on A
        verbose [INPUT] controls chatter
        _extra  [INPUT] use this to pass defined variables to subroutines:-
                ADJUSTIE: TIES, VNAME
                LEVMARQ: JUMPUP, JUMPDN, SVDTHR
                LMCOEFF: FUNCS, SIG
                (note: FUNCS is actually the name of a _procedure_ --
                CURVEFIT is to blame for the confusion)

subroutines
        ADJUSTIE
        LEVMARQ
            LMCOEFF
            SVDC
            SVSOL
```

```
history
        vinay kashyap (Oct98)
        added parameter YFUNC (VK; Dec98)
        bug fix: crashing if freeze not set (VK; 99Aug)
        now returns correct chisq if all params are frozen (VK; JanMM)
        also adjust ERRA for ties (VK; FebMM)
        what if adjusted ERRA becomes 0? (VK; MarMM)
        added keywords TIPTOE and VERBOSE, changed behavior if
          dead end to now tiptoe around point (VK; Nov04)
```

B.59 FITLINES [/fitting]

```
function        fitlines
        widget-based procedure to fit Gaussians and Lorentzians (or
        combinations, or other 3-parameter functions) to lines in a
        spectrum and determine line fluxes.  returns a structure of
        the form
          {POS,PERRP,PERRM,PERRC,$
           FLX,FERRP,FERRM,FERRC,$
           WDT,WERRP,WERRM,WERRC,$
           THAW,TYPE,TIES,EPITHET,CONLEVX,CONLEV,CONSIGX,CONSIG,COMMENT}
        All these are also available as output individually as keywords.
        See keyword descriptions for what the variables mean.

syntax
        fxstr=fitlines(x,y,ysig=ysig,funcs=funcs,/intens,dchi=dchi,$
        pos=pos,perrp=perrp,perrm=perrm,perrc=perrc,$
        flx=flx,ferrp=ferrp,ferrm=ferrm,ferrc=ferrc,$
        wdt=wdt,werrp=werrp,werrm=werrm,werrc=werrc,$
        thaw=thaw,type=type,epithet=epithet,ties=ties,$
        conlev=conlev,consig=consig,comment=comment,$
        histerix=histerix,$
        /dumb,verbose=verbose,xsize=xsize,ysize=ysize,$
        wid=wid,dynrng=dynrng,/posve,/negve,itmax=itmax,chithr=chithr,$
        jumpup=jumpup,jumpdn=jumpdn,svdthr=svdthr,missing=missing,$
        /noday,/notime,/nouser,/nopack,stacol=stacol,stacol=stacol,$
        stathk=stathk,/nuthin)

parameters
        x       [INPUT; required] absissa, e.g., wavelength or energy
        y       [INPUT; required] count spectrum Y(X)
                * size of Y must match that of X

keywords
        ysig    [INPUT] errors on Y
                * default is sqrt(abs(Y)+0.75)+1.
        funcs   [INPUT] name of user defined function (actually a procedure)
                that takes as input X and A (the model parameters), and returns
                Y(X;A) and the partial derivatives dY/dA
                * default is set to X3MODEL
                * NOTE: if MPFIT is chosen as the optimization algorithm,
                  then "_f" is automatically appended to the name.  This
                  is because MPFIT requires a _function_, not a procedure.
                  The program corresponding to X3MODEL is X3MODEL_F() and
                  that corresponding to LIBMODEL is LIBMODEL_F()
```

```
intens  [INPUT] if set, Y(X) is assumed to be intensity, i.e.,
        (counts|ergs|.../bin_unit/...)
        * default is to take Y(X) to be (counts|ergs|.../bin/...)
        * passed straight to FITLINES_EVENT
        * WARNING: this has >not< been road-tested!
dchi    [INPUT] delta-chisq threshold for projection errors
        * default is 1.0
        * may be changed within FITLINES_EVENT, but those changes
          will not be propagated back.
pos     [I/O] best-fit line positions
        * length of vector shows number of components
perrp   [I/O] 1-sided error on POS (POS+PERRP is upper bound)
perrm   [I/O] 1-sided error on POS (POS-PERRM is lower bound)
perrc   [I/O] DCHI threshold used in computing PERR?
flx     [I/O] best-fit fluxes in the lines
ferrp   [I/O] 1-sided error on FLX (FLX+FERRP is upper bound)
ferrm   [I/O] 1-sided error on FLX (FLX-FERRM is lower bound)
ferrc   [I/O] DCHI threshold used in computing FERR?
wdt     [I/O] best-fit widths (sigma, core-radius, etc.) of the lines
werrp   [I/O] 1-sided error on WDT (WDT+WERRP is upper bound)
werrm   [I/O] 1-sided error on WDT (WDT-WERRM is lower bound)
werrc   [I/O] DCHI threshold used in computing WERR?
        * on input, FLX, WDT, and ?ERR? are forced to match the length
          of POS: excess elements are thrown away, and insufficient
          lengths are made up by padding with first elements, 1s, etc.
        * on output ?ERRM are identical to ?ERRP >unless< the
          projected errors have been calculated using ERORS, in
          which case the values may be different
        * on output, places where ?ERRC contain 0's are those where
          the computed errors are 1-sigma formal curvature errors
thaw    [I/O] integer array signaling frozen (0) or thawed parameter (1)
        * refers to sequences of {POS,WDT,FLX} for each component in
          a 3-parameter model (cf. X3MODEL) -- whatever goes for the
          appropriate user-defined function.
        * length must match 3*length(POS)
        * default is to freeze defined input, thaw all new additions
type    [I/O] type of model ('gauss', 'lorentz', etc.)
        * default is 'gauss'
epithet [I/O] label for each component
        * labels, obtained, for example, with IDLABEL
        * Merriam-Webster> a characterizing word or phrase accompanying
          or occurring in place of the name of a person or thing
ties    [I/O] any ties between parameters?
conlev  [I/O] the continuum that was taken out of the spectrum
        * CONLEV must match the size of X and Y else ignored
```

```
        consig  [I/O] error on CONLEV
                * default for CONSIG is sqrt(abs(CONLEV)+0.75)+1.
                * NOTE: CONLEV and CONSIG are compressed using SPLAC in
                  the output that gets returned via the structure.  That
                  structure therefore also has appropriate abscissae
                  CONLEVX and CONSIGX.
        comment [OUTPUT] descriptive string
        histerix        [OUTPUT] a structure containing the state at each step
                        of the fitting process
                        * if explicitly set to 0, will not contain any output
                        * passed w/o comment to FITLINES_EVENT
       oldstr  [INPUT] old fitlines structure from which to start with
        _extra  [INPUT ONLY] use this to pass defined keywords to subroutines
                PICKRANGE: XSIZE, YSIZE, WID, DYNRNG
                LINEREM: POSve, NEGve
                FIT_LEVMAR: ITMAX, CHITHR, DUMB
                LEVMARQ: JUMPUP, JUMPDN, SVDTHR
                MK_3MODEL: MISSING
                MK_SLANT: ANGLE, BETAP
                ERORS: VERBOSE
                STAMPLE: NUTHIN, NODAY, NOTIME, NOUSER, NOPACK, STACOL,
                         STASIZ, STATHK

subroutines
        FITLINES_EVENT
        FMTSTUFF
        PICKRANGE
        FUNCS
        (X3MODEL/LIBMODEL/MPFITFUN)
        (MK_3MODEL, MK_GAUSS, MK_LORENTZ, MK_SLANT, MK_ROGAUSS, MK_POWLAM, MK_POLY1D)
        LINEREM
        SETCONT
        ERORS
        FIT_LEVMAR
        ADJUSTIE
        LEVMARQ
        LMCOEFF
        CURVE_FIT
        SPLAC
        WHICH
        KILROY
        WHEE
        STAMPLE
        PEASECOLR
        IS_KEYWORD_SET
```

```
known bugs
        cannot handle spectra with varying bin sizes
        INTENS has not been tested
        SPLAC keywords are not gracefully handled
        help is available only in UNIX

history
        vinay kashyap (Oct98)
        added renormalization option (VK; Nov98)
        added keywords INTENS,DCHI,PERRL,WERRL,FERRL,PERRC,WERRC,FERRC;
          changed output structure format (VK; FebMM)
        changed keywords PERR,WERR,FERR to PERRU,WERRU,FERRU; also the
          corresponding output structure fields (VK; MarMM)
        changed ?ERRU to ?ERRP and ?ERRL to ?ERRM to avoid confusion
          with ERR[U,L] of ERORS (VK; MarMM)
        moved QUIT to end of row, far from FIT; added ability to save
          to disk; changed ?ERR[M,C] to be I/O; added TIES to output
          structure (VK; JulMM)
        added labeling capability via keyword EPITHET; reorganized
          to combine 6th and 7th rows (VK; AugMM)
        changed location of doc file to ardb ; allowed call to STAMPLE
          (VK; DecMM)
        changed default of ?ERRC to 0.0 (VK; JanMMI)
        per Antonio Maggio's suggestion, removed keyword MULTIPLE from
          call to WIDGET_LIST (VK; FebMMI)
        changed suffix of help file from ".doc" to ".hlp" (VK; FebMMI)
        added code to make it easier to switch between color tables
          (VK; Oct02)
        added monte-carlo errors as option (LL; Aug03)
        added keyword HISTERIX (VK; Feb04)
       added keyword OLDSTR (LL; Jul05)
        updated for IDL5.6 keyword_set([0]) behavior change for vectors
          (VK; 20Mar2006)
```

B.60 FITLINES_EVENT [/fitting]

```
procedure       fitlines_event
        widget event handler subroutine for FITLINES.  see that routine
        for a description of variables, etc.  only rudimentary consistency
        checks are carried out here.  uses subroutine FMTSTUFF, which is
        included in this file.

syntax
        fitlines_event,x,y,pos,wdt,flx,perr,werr,ferr,thaw,type,ties,epithet,$
        conlev,consig,chisq,funcs,sigy,widg, perrm=perrm,werrm=werrm,$
        ferrm=ferrm,perrc=perrc,werrc=werrc,ferrc=ferrc,/intens,rmfstr=rmfstr,$
        histerix=histerix

subroutines
        FMTSTUFF
        PICKRANGE
        FUNCS
        (X3MODEL/LIBMODEL)
        (MK_3MODEL, MK_GAUSS, MK_LORENTZ, MK_SLANT, MK_ROGAUSS, MK_POWLAM, MK_POLY1D)
        LINEREM
        SETCONT
        ERORS
        FIT_LEVMAR , CURVE_FIT , MPFITFUN
        ADJUSTIE
        LEVMARQ
        LMCOEFF
        STAMPLE
        KILROY
        WHEE
        PEASECOLR
        IS_KEYWORD_SET

history
        vinay kashyap (Oct98)
        added renormalization option (VK; Nov98)
        various bug fixes and reorganizations (VK; Dec98)
        now allows fitting a single parameter (VK; Aug99)
        enhancements to include call to ERORS, setting YRANGEs,
          INTENSity units, etc. (VK; FebMM)
        avoid repeat calls to CW_BGROUP, saving tremendous headaches;
          bug correction CONTINUUM->AdjustNorm (VK; MarMM)
        bug correction -- undefined STYPE for new component; rudimentary
          changing parameter labels (VK; JunMM)
        moved QUIT to end of row, far away from FIT; added DUMP to disk
```

```
          (VK; JulMM)
        added error-bar plotting capability via VIEW->SetDefaults;
          pass x,y-range to setcont; added labeling capability via
          keyword EPITHET; merged 6th and 7th rows; bug correction
          with xcont in adjust norm (VK; AugMM)
        changed location of help file to ardb; added calls to STAMPLE
          (VK; DecMM)
        bug fixes: invisible error output, adding model deleted existing
          projected errors, streamlined saves (VK; JanMMI)
        per Antonio Maggio's suggestion, removed keyword MULTIPLE from
          call to WIDGET_LIST; improved color-scale setting for 24-bit
          consoles; replaced call to WHICH by call to ROUTINE_INFO
          (VK; FebMMI)
        various extra info messages (VK; AprMMI)
        changed y to z on lines 1012 and 1285, because.. y was
          inappropriate for some reason? (VK; Jun01)
        was crashing due to missing ZSIG in cont_acpt (VK; Aug01)
        bug was not passing predefined CONLEV to SETCONT (piecewise) (VK; Apr02)
        handle color tables in 24-bit displays (VK; Jun02)
        bug correction -- if all ties were deleted, then TIES was turning into
          and integer array (VK; Jul02)
        made changes to plotting so that hardcopy won't come out with annoying
          blank sheets (VK; Sep02)
        changed default colors to be compatible with PEASECOLOR (VK);
          added hooks into MPFIT (Liwei Lin/VK; Oct02)
        tied the left column of parameter values to changes in the main parameter
          list window on the right (VK; Apr03)
        added monte-carlo errors as option and added keyword RMFSTR (LL; Aug03)
        added keyword HISTERIX (VK; Jul03)
        bug correction: algo_type undefined unless fit is run first (LL; Aug04)
        updated for IDL5.6 keyword_set([0]) behavior change for vectors
          (VK; 20Mar2006)
        modified some widget instructions to be slightly clearer (VK; Aug06)

**********************************************************************

subroutine      fmtstuff
        takes in the model as described, and returns it properly formatted
        for display in the widget.  parameters are named as before.  only
        minimal consistency checks are carried out.
usage
        fmtstuff,pos,wdt,flx,thaw,type,epithet,perr,werr,ferr,$
          perrm=perrm,werrm=werrm,ferrm=ferrm,$
          perrc=perrc,werrc=werrc,ferrc=ferrc,$
          plist=plist,list=list,$
```

```
          tiep=tiep,tiew=tiew,tief=tief,ties=ties,delcmp=delcmp

keywords
        plist   [OUTPUT] appropriately formatted model parameter list
        list    [OUTPUT] denoting frozen/thawed components
        tiep    [I/O] ties specially setting the range on positions
        tiew    [I/O] ties specially setting the range on widths
        tief    [I/O] ties specially setting the range on fluxes
        ties    [I/O] rest of ties
        delcmp  [ACTION] if set to component number, deletes component DELCOMP
                and all TIES that contain references to parameters in that
                component
        _extra  [INPUT] pass defined keywords to STAMPLE (NODAY,NOTIME,
                NOUSER,NOPACK,STACOL)
```

B.61 FITLINES_UNDUMP [/scrypt]

```
script  fitlines_undump
        restores the variables saved to disk from FITLINES_EVENT
        and calls FITLINES again.

warning
        this will restore all saved variables from within FITLINES_EVENT.
        all of your existing variables with the same names will be
        overwritten.  best to always start from a clean environment.

        save files written in IDL 5.4 are incompatible with earlier versions,
        but can be restored under certain circumstances using the CMSVLIB
        package of Craig Markwardt
        (http://cow.physics.wisc.edu/~craigm/idl/idl.html)
        This program has a "restore" command right at the start that
        may fail, but if the variables have already been loaded in,
        just type .SKIP and .CON, and bob's your uncle.

usage
        fitsavfil='fitlines.save'
        .run fitlines_undump

input
        FITSAVFIL       name of IDL save file containing the variables
                        dumped from within the FITLINES GUI

output
        FITSTR  structure containing the fit parameters

history
        vinay kashyap (MMJul)
        cosmetic surgery (VK; DecMM)
        included keyword HISTERIX (VK; FebMMIV)
```

B.62 FLARECLASS [/ssw]

```
function       flareclass
        compute the peak flux of a solar flare based on its GOES classification

        It is assumed that the classification is given as a letter followed by a
        number.  The letter is used to define the intensity level, and the number
        is a multiple of it.  The base levels are:
        A : 1e-8 - 1e-7 W/m^2
        B : 1e-7 - 1e-6 W/m^2
        C : 1e-6 - 1e-5 W/m^2
        M : 1e-5 - 1e-4 W/m^2
        X : >1e-4 W/m^2
        see http://www.spaceweather.com/glossary/flareclasses.html

syntax
        fx=flareclass(class,cgs=0,/atSun,verbose=verbose)

parameters
        class   [INPUT; required] string variable that contains the GOES class

keywords
        cgs     [INPUT] by default, the output is returned in cgs, as ergs/s/cm^2 at Earth
                * set this explicitly to 0 to return the output in SI units
        atSun   [INPUT] if set, multiplies by 4*pi*AU^2 to return peak luminosity of flare
                at the Sun
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

history
        Vinay Kashyap (MMXV.VI)
```

B.63 FLUX_TO_EM [/]

```
function        flux_to_em
        return emission measures (-1 where undeterminable) at various
        temperatures consistent with given line fluxes and line emissivities.

        for each temperature, assume a delta-function emission measure,
        compute line fluxes seen through some instrument, account for
        interstellar absorption, and scale to match the observed flux.

syntax
        em=flux_to_em(emis,flux=flux,logT=logT,wvl=wvl,Z=Z,NH=NH,$
        defEM=defEM,noph=noph,thresh=thresh, abund=abund,/temp,/ikev,$
        effar=effar,wvlar=wvlar,fh2=fh2,he1=he1,heII=heII,/fano)

parameters
        emis    [INPUT; required] 2D array of line emissivities, EMIS(LOGT,WVL)
                * if 1-D, assumed to be EMIS(LOGT)

keywords
        flux    [INPUT] observed fluxes
                * if not given, assumed to be 1 [(ph|erg)/s/cm^2]
                * size is forced to match 2nd dimension of EMIS
                  if <, filled with 1s
                  if >, extra elements are ignored
                * NOTE: the author realizes that there may be multiple IDs
                  for the single observed line, and directs the inquirer
                  to the function ID_TO_EMIS, which returns the appropriately
                  concatenated emissivity table.
        logT    [INPUT] log_10(Temperature [K]) at which EMIS is defined
                * if not given, assumed to go from 4 to 8 in even steps
                * interpolated if size does not match 1st dimension of EMIS
        wvl     [INPUT] wavelengths at which EMIS is defined
                * if not given or if size does not match 2nd dimension of EMIS,
                  (a) interstellar absorption correction is not made
                  (b) all fluxes >gotta be< [ergs/s/cm^2]
        Z       [INPUT] atomic numbers
                * if not given, assumed to be 1s (i.e., H. i.e., no
                  abundance corrections are made)
                * size is forced to match 2nd dimension of EMIS
                  if <, filled with 1s
                  if >, extra elements are ignored
        NH      [INPUT] H column density [cm^-2]
                * ignored if WVL is incorrect
        defEM   [INPUT; default=1e14] default EM to use prior to scaling [cm^-5]
```

```
        noph    [INPUT] if set, FLUX is assumed to be in [ergs/s/cm^2], and
                no attempts are made to convert things to [ph/...]
                * forcibly set if WVL is incorrect
        thresh  [INPUT; default=1e-2] dynamic range in output curves,
                relative to maximum in each curve
        _extra  [INPUT ONLY] use this to pass defined keywords to subroutines
                LINEFLX [ABUND,TEMP,IKEV,EFFAR,WVLAR]
                ISMTAU [FH2,HE1,HEII,FANO]

restrictions
        * EFFAR and WVLAR must be correctly defined, or else the output
          will be garbage
        * requires subroutines
          LINEFLX
              WHEE
          ISMTAU

history
        vinay kashyap (Oct98)
        bug correction re 0 EMIS (VK; Nov98)
        bug correction re 0 NH (VK; AugMM)
```

B.64 FLUXCURVE [/timing]

```
procedure       fluxcurve
        compute a fluxed light curve from low-spectral-resolution
        events data

syntax
        fluxcurve,time,chan,charf,chnrg,fluxlc,tgrid,flxerr,$
        ctlc=ctlc,/adapbin,tmin=tmin,tmax=tmax,tbin=tbin,$
        tstart=tstart,tstop=tstop,/shift1,verbose=verbose,$
        /slowok,snrthr=snrthr

parameters
        time    [INPUT; required] photon arrival times
        chan    [INPUT; required] the PI or PHA values of each photon
        charf   [INPUT; required] the average effective area in each channel
        chnrg   [INPUT; required] the average photon energy in each channel
        fluxlc  [OUTPUT; required] the fluxed light curve, in [ergs/s/cm^2]
        tgrid   [OUTPUT; required] the bin boundaries over which the light
                curve is computed
        flxerr  [OUTPUT; optional] the standard errors on the fluxed light
                curve, computed as stddev(fluxes)/nphotons in each bin

keywords
        ctlc    [OUTPUT] the counts light curve, in [ct/s]
        adapbin [INPUT] if set, rebins adaptively to ensure a
                minimum number of counts in each bin, by pushing
                the counts histogram through SMOOTHIE
                * assumes errors to be sqrt(CTLC*TBIN)
                * for each group, the fluxes from the individual
                  bins are averaged
        tmin    [INPUT] the minimum time value to consider
                * default: min(TIME) < min(TSTART)
        tmax    [INPUT] the maximum time value to consider
                * default: max(TIME) > max(TSTOP)
        tbin    [INPUT] the bin size
                * default: TMAX-TMIN
        tstart  [INPUT] array of start times of GTI's
        tstop   [INPUT] array of stop times of GTI's
        shift1  [INPUT] if set, assumes that the channel numbers
                start from 1, not from 0
                * see the FIRSTCHAN field in the response matrix
                  structure read in via RD_OGIP_RMF()
        verbose [INPUT] controls chatter
        _extra  [INPUT ONLY] pass defined keywords to subroutines
```

```
                SMOOTHIE : SNRTHR
                REBINW : SLOWOK

subroutines
        TI_COVER
        REBINW
        FINDEX
        SMOOTHIE

history
        vinay kashyap (MMVI.IX)
```

B.65 FOLD_IONEQ [/]

```
function        fold_ioneq
        folds in ionization equilibrium fractions of the various ions
        of a given element with line emissivities (which include only
        level population info; see RD_LINE()).  returns
        emis(T;Z,ion)*ioneq(T;Z,ion)

syntax
        lflx=fold_ioneq(emis,Z,ion,logt=logt,tmax=tmax,trng=trng,$
        level=level,userarr=userarr,chifil=chifil,chidir=chidir,$
        eqfile=eqfile,verbose=v)

parameters
        emis    [INPUT; required] 1- or 2-D array of line emissivities as
                a function of temperature
                * if 2D, assumed to be EMIS(Temperature,Wavelength)
                * if 1D, assumed to be EMIS(Temperature), unless the
                  number of elements matches that of Z
        Z       [INPUT; required] atomic number(s) corresponding to
                each wavelength
                * if size does not match the 2nd dimension of EMIS, then
                  if >, ignore extra elements
                  if <, but >0, use available ones, and use Z[0] for rest
        ion     [INPUT; default: Z+1] ionic state(s) corresponding to
                each wavelength
                * if size does not match the 2nd dimension of EMIS, then
                  if >, ignore extra elements
                  if <, but >0, use available ones, and use ION[0] for rest
                * NOTE: this is the ionic state >>that matters<<, i.e.,
                  the ionic state of the species that populates the
                  upper level.  See keyword JON of RD_LINE, which is
                  what is needed here.

keywords
        logt    [INPUT] 1D array of log10(temperature [K]) at which EMIS
                is given. (default: findgen(81)*0.05+4)
                * if size does not match that of EMIS, use default
        level   [INPUT; default: 0.5] level at which to determine the
                temperature range on line flux (in other words, default
                will be to return the FWHM)
                * if <1, TRNG=LOGT(where(FLUX>LEVEL*MAX(FLUX))
                * if >=1, TRNG=LOGT(where(FLUX>(LEVEL/100.)*MAX(FLUX))
                * if <=0 or >=100, set to 0.5
        tmax    [OUTPUT] log10(temperature) at which contribution is maximum
```

```
        trng    [OUTPUT] range in LOGT (LEVEL of MAX)
                -- TRNG(0,*) is lower bound, TRNG(1,*) is upper bound
        userarr [INPUT] an array of ionization fractions in the same
                format as is returned by RD_IONEQ(), and has the size
                [n(T),max(Z)+1,n(Z)]
                * if this is given and is legit (i.e., matches the
                  supplied EMIS and LOGT), then EQFILE and CHIFIL are
                  ignored
        eqfile  [I/O] file from which to read in ion-balance data
                * default is to read in the CHIANTI file, !IONEQF
                * hard-coded default is
                  ioneq/chianti.ioneq
                * if not given, but CHIFIL is given, uses that.
        chifil  [I/O] if set, reads in from CHIANTI-type database; must be
                set to the name of the file containing the ionization
                equilibrium info (i.e., CHIDIR/CHIFIL)
                * may be overridden with EQFILE
                * default is ioneq/chianti.ioneq
                * currently, set automatically if not given.
        verbose [INPUT] higher the number, greater the chatter
        _extra  [INPUT] use to transmit defined keywords to called functions
                * RD_IONEQ [CHIDIR]

subroutines
        -- WHEE
        -- SETSYSVAL
        -- RD_IONEQ [READ_IONEQ (%)]
        (%) CHIANTI subroutine, used as is

history
        vinay kashyap (Dec96)
        removed call to KILROY and added call to WHEE, hacked default
          use of READ_IONEQ until more options become available; corrected
          bug that was reading in ionstate-1 rather than ionstate; added
          code to handle Z=0 (VK; Feb97)
        bug: if Z is scalar take em all to be same element (VK; Apr97)
        added keyword VERBOSE (VK; MarMM)
        streamlined meanings of EQFILE, CHIFIL (VK; DecMM)
        changed default for EQFILE; now pass VERBOSE to RD_IONEQ; added
          call to SETSYSVAL (VK; Jul01)
        allowed EQFILE to be "none" (VK; Feb04)
        changed default behavior of Z and ION when sizes don't match EMIS;
          now uses first element of input if given (VK; Nov04)
        bug correction, ION is offset by 1 from array index (LL/VK; Dec04)
        added keyword USERARR, slightly modified behavior when EMIS
```

```
  was input as 1D (VK; Aug08)
changed default of EQFILE and CHIFIL to ioneq/chianti.ioneq (VK; Aug13)
```

B.66 FPUFIX [/external]

```
NAME:
      FPUFIX

PURPOSE:

      This is a utility routine to examine a variable and fix problems
      that will create floating point underflow errors.

AUTHOR:

      FANNING SOFTWARE CONSULTING
      David Fanning, Ph.D.
      1645 Sheely Drive
      Fort Collins, CO 80526 USA
      Phone: 970-221-0438
      E-mail: davidf@dfanning.com
      Coyote's Guide to IDL Programming: http://www.dfanning.com

CATEGORY:

      Utilities

CALLING SEQUENCE:

      fixedData = FPUFIX(data)

ARGUMENTS:

      data :         A numerical variable to be checked for values that will cause
                     floating point underflow errors. Suspect values are set to 0.

KEYWORDS:

      None.

RETURN VALUE:

      fixedData:    The output is the same as the input, except that any values that
                    will cause subsequent floating point underflow errors are set to 0.

COMMON BLOCKS:
      None.
```

```
EXAMPLES:

      data = FPTFIX(data)

RESTRICTIONS:

    None.

MODIFICATION HISTORY:

      Written by David W. Fanning, from Mati Meron's example FPU_FIX. Mati's
         program is more robust that this (ftp://cars3.uchicago.edu/midl/),
         but this serves my needs and doesn't require other programs from
         Mati's library.  24 February 2006.
```

B.67 FREEBOUND_CH [/external]

```
PROJECT:  CHIANTI

     CHIANTI is an Atomic Database Package for Spectroscopic Diagnostics of
     Astrophysical Plasmas. It is a collaborative project involving the Naval
     Research Laboratory (USA), the University of Florence (Italy), the
     University of Cambridge and the Rutherford Appleton Laboratory (UK).

NAME

     FREEBOUND

PURPOSE:

     Calculates the free-bound (radiative recombination) continuum.

     A report giving detailed information on how the CHIANTI
     free-bound is calculated is available at:

     http://solar.bnsc.rl.ac.uk/~young/chianti/freebound.pdf

     To calculate the continuum spectrum the user should specify
     whether the plasma structure is specified through a differential
     emission measure (DEM) curve or through emission measure (EM)
     values. For the former, the DEM values are specified through the
     DEM_INT keyword and for the latter the EM values are specified
     through the EM_INT keyword. If neither are specified EM_INT
     values of unity are assumed.

     If DEM_INT values are specifed they must be specified on a
     temperature scale that has constant intervals in log10(T). EM_INT
     values do not have this restriction.

     Note that EM_INT must be used for isothermal plasmas.

INPUTS

     TEMP    Temperature in K (can be an array). If the keyword
             /DT is used, the
             temperatures must be spaced at equal intervals of
             log10(T). E.g., log T=6.0, 6.1, 6.2, 6.3, ...

     WVL     Wavelength in angstroms (can be an array). If /keV is
             set, then WVL is assumed to contain energies in keV units.
```

```
OUTPUTS

     INT     Free-bound continuum intensity in units of
             10^-40 erg cm^3/s/sr/Angstrom
             ( integral(N_H N_e dh) in cm^-5) if a DEM is not defined.

             If DEM values are defined, it is assumed that they are given
             as N_H N_e dh/dT.  The units are 10^-40 erg/cm^2/s/srAngstrom

             If T is given as a 1-D array, then the output will be a
             2-D array, with one element for each temperature and
             wavelength (but also see SUMT).

             If the keyword /keV is set, then the units of INT will be
             10^-40 erg cm^3/s/sr/keV.

OPTIONAL INPUTS

   DEM_INT   This should be the same size as TEMP and contain
             differential emission measure values. Specifying
             DEM_INT places a restriction on TEMP (see above). Note
             that DEM_INT values are multiplied by the factor dT
             within the routine, whereas EM_INT values are not.

   EM_INT    This should be the same size as TEMP and contain
             emission measure values. Note
             that DEM_INT values are multiplied by the factor dT
             within the routine, whereas EM_INT values are not. If
             neither EM_INT or DEM_INT are specified, then a EM_INT
             vector of same size as TEMP with values of unity is
             assumed.

     IZ     Only calculate continuum for the element with atomic
            number IZ

     ION    (To be used in conjunction with IZ.) Calculated continuum
            for a single ion (IZ, ION).

KEYWORDS

     NO_SETUP If the procedure setup_elements has already been called
              then the keyword /nosetup should be set to avoid
              repeating this step
```

```
     MIN_ABUND If set, calculates the continuum only from those
               elements which have an abundance greater than
               min_abund.  Can speed up the calculations.  For
               example:
                  abundance (H)  = 1.
                  abundance (He) = 0.085
                  abundance (C)  = 3.3e-4
                  abundance (Si) = 3.3e-5
                  abundance (Fe) = 3.9e-5

     PHOTONS  The output spectrum is given in photon units rather
              than ergs.

     SUMT     When a set of temperatures is given to FREEBOUND, the
              default is to output INTENSITY as an array of size
              (nwvl x nT). With this keyword set, a summation over
              the temperatures is performed.

     VERBOSE  Output information from FREEBOUND.

     KEV      If set, then WVL is assumed to contain energies in keV units
              rather than wavelengths in Angstroms.

COMMON BLOCKS

     ELEMENTS

CALLS

     FREEBOUND_ION, SETUP_ELEMENTS, READ_KLGFB, GET_IEQ

HISTORY

     Ver.1, 24-Jul-2002, Peter Young

     Ver.2, 26-Jul-2002, Peter Young
          revised call to freebound_ion; corrected ion fraction problem

      V 3, 25-May-2005, GDZ
                 corrected routine header.

     Ver.4, 10-Mar-2006, Peter Young
          added keV keyword

     Ver.5, 8-Sep-2009, Peter Young
```

```
          routine now checks to make sure temperatures are evenly
          spaced when the DEM_INT keyword is used; if an error is
          found due to incorrect temperature specificiations, then
          the continuum intensity is returned as zero.

     Ver.6, 6-Oct-2009, Peter Young
           Introduced EM_INT keyword for specifying emission measure
           values for isothermal plasmas.

VERSION     :  6, 6-Oct-2009
     Ver.PoA, 10-Aug-2010, Vinay Kashyap,
         ala changes 18-Feb-2005, LiWei Lin
         Commented common block out and
         Added _extra keyword
         Rename routine freebound_ch
```

B.68 FREEBOUND_ION [/external]

PROJECT: CHIANTI

CHIANTI is an Atomic Database Package for Spectroscopic Diagnostics of Astrophysical Plasmas. It is a collaborative project involving the Naval Research Laboratory (USA), the University of Florence (Italy), the University of Cambridge and the Rutherford Appleton Laboratory (UK).

NAME

FREEBOUND

EXPLANATION

Calculates the free-bound (radiative recombination) continuum from a single ion. Note that the output does not contain the ion fraction, element abundance or differential emission measure term.

INPUTS

TEMP	Temperature in K (can be an array).
WVL	Wavelength in angstroms (can be an array). If the keyword /keV is set, then wvl should instead be given in units of keV.
IZ	Atomic number of ion (e.g., 26 = Fe)
ION	Spectroscopic number of ion (e.g., 13 = XIII)

OUTPUTS

INT	Free-bound continuum intensity. Needs to be multiplied by element abundance, ion fraction and DEM to obtain the final continuum intensity.

OPTIONAL INPUTS

IP	The ionization potential of the ion.
VDATA	An array containing the Verner & Yakovlev data array.
PE	An array containing the PE data from READ_FBLVL

```
     KLGFB   An array containing the KLGFB data from READ_FBLVL

     [Note: the above 3 inputs are used when calling freebound_ion from
      freebound]

KEYWORDS

     NOVERNER  If set, then the Verner & Yakovlev cross-sections will
               not be used.

     KEV      If set, then WVL is assumed to contain energies in keV units
              rather than wavelengths in Angstroms.

COMMON BLOCKS

     None.

CALLS

     READ_FBLVL, ZION2FILENAME, VERNER_XS, KARZAS_XS, CONCAT_DIR,
     READ_IP, READ_KLGFB, FILE_EXIST

PROGRAMMING NOTES

     The way I treat the exponential function in the expression for the
     emissivity may seem strange, but it saves a bit of time in the
     calculation. Basically calculating exp(E-IP+E_i/T) for each level
     was time consuming so I split it into exp(E-IP/T)*exp(E_i/T). The
     first term comes out of the for loop, while the second term is the
     exponential of a vector rather than an array. This made a ~30%
     time-saving for freebound.

HISTORY

     Ver.1, 24-Jul-2002, Peter Young

     Ver.2, 26-Jul-2002, Peter Young
         Added /noverner keyword.

     Ver.3, 30-Jul-2002, Peter Young
         Speeded up routine by modifying treatment of exponential.

     Ver.4, 10-Mar-2006, Peter Young
         Added /keV keyword
```

```
Ver. 5 8-Jun-2010, Ken Dere
    added check for n_params()
```

B.69 FREEFREE_CH [/external]

```
PROJECT     :  CHIANTI

      CHIANTI is an Atomic Database Package for Spectroscopic Diagnostics of
      Astrophysical Plasmas. It is a collaborative project involving the Naval
      Research Laboratory (USA), the University of Florence (Italy), the
      University of Cambridge and the Rutherford Appleton Laboratory (UK).

NAME

    FREEFREE

 PURPOSE:

    This routine computes the free-free continuum (bremsstrahlung)
    using the fitting formulae of Itoh et al. (ApJS 128, 125, 2000)
    and Sutherland (MNRAS 300, 321, 1998).

    The Itoh et al. data are valid for smaller ranges for temperature
    and wavelength than Sutherland and so for points outside of their
    ranges we use the data of Sutherland.

INPUTS

   TEMP    Temperature (in K). They do not have to be in monotonically
           increasing order.

   WVL     Wavelengths in angstroms. Can be a scalar or vector. If /keV is
           set, then WVL is assumed to contain energies in keV units.

OUTPUTS

   INT     Free-free continuum intensity in units of
           10^-40 erg cm^3/s/sr/Angstrom
           [ integral(N_H N_e dh) in cm^-5 ] if a DEM is not defined.

           If DEM values are defined, it is assumed that they are given
           as N_H N_e dh/dT.  The units are 10^-40 erg/cm^2/s/sr/Angstrom.

           If T is given as a 1-D array, then the output will be a 2-D array,
           with one element for each temperature and wavelength
           (but also see SUMT).

           If the keyword /keV is set, then the units of INT will be
```

```
           10^-40 erg cm^3/s/sr/keV

OPTIONAL INPUTS

   DEM_INT   This should be the same size as TEMP and contain
             differential emission measure values. Specifying
             DEM_INT places a restriction on TEMP (see above). Note
             that DEM_INT values are multiplied by the factor dT
             within the routine, whereas EM_INT values are not.

   EM_INT    This should be the same size as TEMP and contain
             emission measure values. Note
             that DEM_INT values are multiplied by the factor dT
             within the routine, whereas EM_INT values are not. If
             neither EM_INT or DEM_INT are specified, then a EM_INT
             vector of same size as TEMP with values of unity is
             assumed.

   MIN_ABUND This keyword allows the specification of a minimum abundance,
             such that any elements with an abundance (relative to
             hydrogen) less than MIN_ABUND will not be included in the
             calculation. E.g., MIN_ABUND=1e-5.

KEYWORDS

   NO_SETUP By default the routine asks the user which ion balance
            and abundance files to use via pop-up widgets. If
            /no_setup is used then this data is taken from the common
            block.

   SUMT     The default is to output the intensity array as an array
            of size (nwvl x nT). Setting this keyword performs a sum
            over the temperatures to yield a vector of same size as
            the input wavelengths, thus producing the complete
            free-free spectrum.

   PHOTONS  Gives output emissivity in photon units rather than ergs.

   KEV      If set, then WVL is assumed to contain energies in keV units
            rather than wavelengths in Angstroms.

CALLS

   SUTHERLAND_CH, ITOH_CH
```

```
COMMON BLOCKS

   NONE (ELEMENTS commented out)

PROGRAMMING NOTES

   The Itoh fitting formula is only valid for (6.0 LE logT LE 8.5).
   For temperatures below this, we thus switch to the Sutherland
   fitting formula. There is very little (<1%) between the two at
   logT=6.

   Itoh also has a constraint on the quantity u=hc/kTl (l=wavelength),
   such that (-4 LE log u LE 1.0). The upper limit corresponds to the
   continuum being cut-off prematurely at low wavelengths. E.g., for
   T=10^6 the cutoff is at 14.39 angstroms. For these low wavelengths
   we also use the Sutherland data to complete the continuum. Note that
   the continuum at these wavelengths is very weak

MODIFICATION HISTORY

   Ver.1, 5-Dec-2001, Peter Young
        Completely revised to call the separate itoh.pro and
        sutherland.pro routines.

   V. 2, 21-May-2002,  Giulio Del Zanna (GDZ),
         Corrected the description of the  units.
         Added verbose keyword and a printout.

   V. 3, 22-May-2002,  Peter Young (PRY)
         Added MIN_ABUND optional input.
         Changed ioneq_t to ioneq_logt (GDZ).

   V 4, 25-May-2005, GDZ
                 corrected routine header.

   V. 5, 9-Mar-2006, Peter Young
         Added /kev keyword

   V. 6, 9-May-2008, Peter Young & Kevin Tritz
         Modifed code so that temperatures can be specified in a unordered
         list.

   V. 7, 7-Oct-2009, Peter Young
         Added EM_INT= keyword; simplified how the routine decides
         which calculation to use (Itoh or Sutherland)
```

```
VERSION     :  7, 7-Oct-2009

   V.PoA, 10-Aug-2010, Vinay Kashyap
          a la 18-Feb-2005, LiWei Lin
         Commented common block out and
         Added _extra keyword
         Rename routine freefree_ch
         Edit calls to sutherland and itoh to sutherland_ch and itoh_ch
```

B.70 GENERATIO [/util]

```
procedure       generatio
        a generalized mechanism to compute flux ratios and errors
        from given input fluxes and a well-defined definition of
        how the ratios are constructed

        all of the input fluxes are assumed to be independent
        of each other and uncorrelated

syntax
        generatio,fx,rcode,rx,fxerr=fxerr,rxerr=rxerr,verbose=v,$
        dfx_mul=dfx_mul,dfx_add=dfx_add

parameters
        fx      [INPUT; required] input fluxes
        rcode   [INPUT] string array describing which of the
                input fluxes must be considered only as ratios
                * basic format is: "sP#[,sP#[,...]]" where
                  -- "#" is an integer flag describing the ratio being
                     constructed
                  -- "P" is a positional descriptor and can take on
                     values N (for numerator) or D (for denominator)
                  -- "s" stands for the sign with which the flux
                     goes into the ratio "+" or "-"
                * e.g., to construct a simple ratio FX[2]/FX[1],
                  RCODE=[’’,’+D1’,’+N1’]
                * e.g., to construct two hardness ratios
                        FX[3]/FX[1] and
                        (FX[3]-FX[1])/(FX[3]+FX[1]),
                  RCODE=[’’,’+D1,-N2,+D2’,’’,’+N1,+N2,+D2’]
                * if size is incompatible with FX, then no action
                  is taken.
        rx      [OUTPUT] array of ratios constructed using FX and RCODE

keywords
        fxerr   [INPUT] 1-sigma errors on FX
                * size must match that of FX.  otherwise,
                  -- if single-element, assumed to represent
                     -- a fractional error if >0 and <1
                     -- a percentage error if >1 and <100
                     -- an abs(constant) error if >100 or <0
                  -- ignored otherwise
        rxerr   [OUTPUT] 1-sigma errors on RX, computed only if FXERR
                is given and is legal
```

```
        verbose [INPUT] controls chatter
        dfx_mul [INPUT] multiplicative factor by which to jiggle FX
                while computing partial derivatives to propagate
                errors (default is 0.05)
                * if scalar, taken to be same factor for all FX
                * if vector, must be of same size as FX
        dfx_add [INPUT] additive factor by which to jiggle FX
                while computing partial derivatives to propagate
                errors (default is 0.05*FXERR)
                * if scalar, taken to be same offset for all FX
                * if vector, must be of same size as FX
        _extra  [JUNK] here only to avoid crashing the program

example
        ;this makes the ratios FX[0]/FX[1] and (FX[1]-FX[0])/(FX[1]+FX[0])
        fx=[10.,5.] & fxerr=0.1 & rcode=['+N1,-N2,+D2','+D1,+N2,+D2']
        generatio,fx,rcode,rx,fxerr=fxerr,rxerr=rxerr
        for i=0,n_elements(rx)-1 do print,rx[i],' +- ',rxerr[i]

        Warning: The numeral suffix in the ratio specification is
        only as a placeholder to determine uniqueness and does _not_
        translate to the index in the output.  For example, try:
        fx=[1.,2.,1.,3.,1.,4.,1.,5.,1.,6.]
        rcode=['+D1','+N1','+D2','+N2','+N3','+D3','+N4','+D4','+D5','+N5']
        generatio,fx,rcode,rx,verbose=100
        print,rcode & print,rx
        generatio,fx,reverse(rcode),rx_reverse,verbose=100
        print,reverse(rcode) & print,rx_reverse

subroutines
        IS_KEYWORD_SET

history
        vinay kashyap (Nov'02)
        cosmetic changes (VK; Dec'02)
        won't spit out error messages if RCODE is 'X', though the FX
          corresponding to that is ignored (VK; Apr'03)
        added warning about ratio sequence -- it's a feature, not a bug;
          corrected bug in case of sequence number exceeding 9 (VK; Dec'03)
        now lets specifying just numerator or just denominator (VK,LL; Jun'04)
        updated for IDL5.6 keyword_set([0]) behavior change for vectors
          (VK; 20Mar2006)
        changed DFX_ADD and DFX_MUL to be arrays of same size as FX
```

B.71 GET_DELIM [/external]

```
 NAME:
        get_delim
 PURPOSE:
        returns file delimiter that is appropriate to VMS or UNIX
 CALLING SEQUENCE:
        delim=get_delim()
 INPUTS:
        none
 OUTPUTS:
       delim=':' if VMS, '/' otherwise
 PROCEDURE:
        checks !version.os system variable
 MODIFICATION HISTORY:
       Written DMZ (ARC) May 1992
       Modified DMZ(SAC) Sept 1997 - added Windows

unction get_delim

n_error,1

s=strlowcase(os_family())

ase os of
'vms'    : delim=':'
'windows': delim='\'
else     : delim='/'
ndcase

eturn,delim & end
```

B.72 GET_IEQ [/external]

```
PROJECT:  CHIANTI

     CHIANTI is an Atomic Database Package for Spectroscopic Diagnostics of
     Astrophysical Plasmas. It is a collaborative project involving the Naval
     Research Laboratory (USA), the University of Florence (Italy), the
     University of Cambridge and the Rutherford Appleton Laboratory (UK).

NAME

     GET_IEQ()

EXPLANATION

     For a specified ion (IZ, ION) and set of temperatures (TEMP) this
     routine takes the ion fraction values tabulated in one of the CHIANTI
     .IONEQ files, interpolates and extracts the values of the ion
     fraction at the input temperatures.

INPUTS

     TEMP   The temperature(s) at which the ion fractions are required.

     IZ     The atomic number of the element (e.g., 26 = iron).

     ION    The spectroscopic number of the ion (e.g., 13 = XIII).

OPTIONAL INPUTS

     IONEQ_LOGT  The temperature output from the READ_IONEQ routine.

     IONEQ_FRAC  The ion fractions from the READ_IONEQ routine.

OUTPUT

     A vector of same length as the input TEMP containing the ion
     fractions at these temperatures.

CALLS

     READ_IONEQ

HISTORY
```

```
     Ver.1, 24-Jul-2002, Peter Young
     v.2, 14-Sept-2005, GDZ
     modified error message.


VERSION 2 14-Sept-2005
```

B.73 GET_IONLIST [/mkemis]

```
function        get_ionlist
        return a list of all the ions for a particular element that
        CHIANTI has data for

syntax
        all_ions=get_ionlist(Z,dielec=dielec,chidir=chianti_topdir)

parameters
        z       [INPUT; required] atomic number
                * must be a scalar integer!

keywords
        chidir  [INPUT] path to CHIANTI top directory
                [default: /data/fubar/SCAR/CHIANTI/dbase]
        dielec  [OUTPUT] integer array of same size as output, specifying
                whether given ion refers to dielectronic recombination
                lines (1) or not (0)
        _extra  [JUNK] here only to prevent crashing program

requires subroutines
        SYMB2ZION
        IS_KEYWORD_SET

history
        vinay kashyap (Nov96; based on CHIANTI's SYNTHETIC.PRO)
        added keyword DIELEC, modified to include CHIANTIv3 dielectronic
          recombination directories (VK; OctMM)
        handle trailing whitespace in masterlist.ions (VK; Jun02)
        updated for IDL5.6 keyword_set([0]) behavior change for vectors
          (VK; 20Mar2006)
```

B.74 GET_PIMMS_FILE [/util]

```
function        get_pimms_file
        returns a string containing the name of the file containing the
        effective areas for the given instrument.

usage
        filnam=get_pimms_file(mission,instr,config,special=special,pdir=pdir)

parameters
        mission [INPUT; required] e.g., 'ROSAT', 'AXAF', etc.
        instr   [INPUT] instrument (e.g., 'PSPC', 'ACIS', etc.)
        config  [INPUT] instrument configuration (e.g., 'OPEN', 'BI', etc.)

keywords
        special [INPUT] any special requests
                * e.g., if MISSION='rosat', INSTR='pspc', and CONFIG='R4'
                and SPECIAL='R7', the output will be 'rosat_pspc_r4tor7.area'
        pdir    [INPUT; default='/data/fubar/SCAR/ardb/pimms/data/'] directory
                containing the PIMMS data files

history
        vinay kashyap (Mar98)
        changed PDIR default from /soft/prop_cli/config/pimms/data to
          /soft/pimms/data (VK; Feb03)
        changed PDIR default from /soft/pimms/data to
          /soft/ciao/config/pimms/data/ (VK; Mar06)
        changed PDIR default from /soft/ciao/config/pimms/data to
          !ARDB/pimms/data/ (VK; Mar13)
```

B.75 GETABUND [/util]

```
function        getabund
                returns elemental abundances in an array

syntax
        abund=getabund(hint,elem=elem,source=source,norm=norm,$
        metals=metals,fipbias=fipbias,abunde=abunde)

parameters
        hint    [INPUT] some hint as to what output is required.  examples:
                'help' -- prints out usage
                'angr', 'anders', 'anders & grevesse', '1989', etc. -- Anders & Grevesse (1989)
                'meyer', '1985', etc. -- Meyer (1985)
                'allen', 'cosmic', '1973' -- Allen (1973)
                'ross', 'aller', 'ross & aller', '1976' etc. -- Ross & Aller (1976)
                'grevesse', 'grevesse et al.', etc. -- Grevesse et al (1992)
                        modifications to Anders & Grevesse (1989)
                'feld', 'feldman', '1992', etc. -- Feldman et al (1992)
                'waljeski', '1994', etc. -- Waljeski et al. (1994)
                'murphy', 'chromospheric', etc. -- Murphy (1985) chromospheric
                'grevesse & anders', '1991' -- Grevesse & Anders (1991)
                'grsa', 'grevesse & sauval', '1998' -- Grevesse & Sauval (1998) "standard
                        abundances"
                'young', 'photo', '1997' -- photospheric abundances from Young et
                        al. (1997)
                'fludra', 'fludra & schmelz', 'hybrid', '1999' -- Fludra & Schmelz (1999) "hybrid
                'asplund', '2005' -- Asplund, Grevesse, & Sauval (2005) 3D hydro
                        (NOTE: even though it has a different reference, is identical to aspl)
                'schmelz', 'schmelz et al', 'coronal', '2012' -- Schmelz et al (2012) "recommended
                'aspl', 'asplund 2009', 'asp ... & scott', '2009', etc. -- Asplund, Grevesse, Sauv
                        model of solar atmosphere to constrain convection
                'aneb', 'anders & ebihara', 'ebihara', '1982', etc. - Anders & Ebihara (1982)
                'wilm', 'wilms, allen, mccray', '2000', etc. -- Wilms, Allen, & McCray (2000)
                'lodd', 'lodders', '2003', etc. -- Lodders (2003)
                'path/file', etc. -- if string contains a "/" in it, then
                        read from disk file which contains one entry per
                        line, with atomic symbol first, and abundance next

keywords
        elem    [INPUT] element -- if given, return abundances for only
                the specified element
                * if string, converts to atomic number using SYMB2ZION
                * if byte, integer, or float, assumed to be atomic number
                * may be an array
```

```
source  [INPUT] another way to say "hint"
        1: Anders & Grevesse (1989)
        2: Meyer (1985)
        3: Allen (1973)
        4: Ross & Aller (1976)
        5: Grevesse et al. (1992)
        6: Feldman et al. (1992)
        7: Waljeski et al. (1994)
        8: Murphy (1985)
        9: Grevesse & Anders (1991)
        10: Grevesse & Sauval (1998)
        11: Young et al. (1997)
        12: Fludra & Schmelz (1999)
        13: Asplund, Grevesse, & Sauval (2005)
        14: Schmelz et al. (2012)
        15: Asplund, Grevesse, Sauval, & Scott (2009)
        16: Anders & Ebihara (1982)
        17: Wilms, Allen, & McCray (2000)
        18: Lodders (2003)
        * overrides HINT iff HINT is not understood
norm    [INPUT] normalization
        * use this to set output format
          NORM=1: output will be n(Z)/n(H) (this is the default)
          NORM>1: output will be NORM(0)+log10(n(Z)/n(H))
                (e.g., NORM=12 is the standard log form)
          0<NORM<1: output will be NORM(0)*n(Z)/n(H)
                (this can be useful when ELEM is set)
          -1<=NORM<0: output will be n(Z)/DEFAULT_n(Z)
                (if you want to compare different abundance tables)
          NORM<-1: output will be log10(n(Z)/DEFAULT_n(Z))
                (I have no idea why anyone would use this)
metals  [INPUT] metallicity.  if set, modifies the abundances of
        all elements past He by the appropriate amount
        * if not set, makes no changes
        * assumed to be in log10 form if
          -- abs(NORM) > 1, or
          -- < 0
        * assumed to be multiplicative factor otherwise
        * NOTE: for example, if norm=12 then metals=0 means
          Solar abundances, while if norm=1 then metals=0
          means complete metal depletion!
fipbias [INPUT] if set to a non-zero scalar, multiplies the
        abundances of low-FIP elements by this factor.  be
        careful not to use it for abundance lists that already
        include the FIP effect.
```

```
                * if not set, or set to 0, does nothing.
        abunde  [OUTPUT] errors on abundances

        _extra  [INPUT] junk -- here only to prevent program from crashing

subroutines
        RDABUND
        SYZE
        SYMB2ZION

history
        vinay kashyap (Dec96)
        avoid analyzing HINT if SOURCE is set (VK; Jan97)
        added 'angr' and tightened 'Anders&Grevesse' v/s 'Grevesse' (VK; Jul97)
        added option of reading from disk file (VK; Jun98)
        added Murphy (1985) data, and allowed ELEM to be array (VK; Jul98)
        corrected bug associated with non-array ELEMs (VK; Sep98)
        added keyword METALS, deleted keywords ALTER, FORM, removed call to
          SETABUND, added call to RDABUND (VK; Dec98)
        added Grevesse & Anders (1991), Grevesse & Sauval (1998), Young et
          al (1998) (VK; Apr99)
        added keyword FIPBIAS (VK; Jul01)
        added Fludra & Schmelz (1999); now HINT overrides SOURCE and program
          warns if there is a conflict (VK; Jun02)
        corrected Meyer abundances for O and Na (VK; Apr04)
        corrected Fludra & Schmelz abundances (VK; Jun04)
        added Asplund, Grevesse, & Sauval abundances; added keyword ABUNDE
          (VK; Apr05)
        B was off by 2 orders of magnitude (thx Marc Audard; VK Jul06)
        streamlined input HINTs; added Schmelz et al. (2012); updated
          bracket notation to IDL5+ (VK Jul12)
        added 'aspl', 'aneb', 'wilm', and 'lodd' from the elemental abundance
          table used in XSPEC plasma code models, as given in ahelp get_xsabund;
          changed convection model from Asplund et al (2005) to aspl; streamlined
          selection keywords (VK Feb13)
        changed the behavior of default to accept !ABREF if defined (VK May28)
```

B.76 GETLOCMAX [/util]

```
function        getlocmax
        returns the position indices where y(x) is a local maximum

syntax
        ilm=getlocmax(x,y,width=width,sigma=sigma,nsigma=nsigma)

parameters
        x       [INPUT; required] abscissae where input function is defined
        y       [INPUT] input function.  if not given, X is taken to be Y
                and array position indices are taken to be X

keywords
        width   [INPUT; default: 1] window in which to search for local
                maxima -- window width = 2*abs(WIDTH)+1 or 3, whichever
                is greater.
        sigma   [INPUT; default: 1+sqrt(abs(Y)+0.75)] error at each point
                * if scalar and NSIGMA is not defined, then SIGMA->NSIGMA
        nsigma  [INPUT; default: 1] multiple of SIGMA to consider as a
                threshold filter
                * to use var(y) as a constant value of SIGMA, do
                  ilm=getlocmax(x,y,sigma=sqrt((moment(y))(1)),/nsigma)

        _extra  [INPUT] junk -- here only to keep routine from crashing

history
        vinay kashyap (Dec96)
        corrected bug with 1-, 2-, or 3-element inputs (VK; Nov98)
```

B.77 GETPOADEF [/]

```
function       getpoadef
        returns the default value for a common PINTofALE variable
        usually for filenames, which are often dependent on where
        the files are installed.

        The way this works is, if a system variable is already set,
        the default is taken from that.  If not, after figuring out
        what the default is (usually by checking where this program
        is located), the appropriate system variable is set to it.

syntax
        default=getpoadef(varname,/force,verbose=verbose)

parameters
        var     [INPUT] name of variable as a character string
                * may be an array of strings
                * special variable names are the only ones recognized
                * if 'help' or '?', lists all the variables for which
                  defaults can be set
                * in some cases default might be a number, but if the
                  input is an array, then the output will always be a
                  string array

keywords
        force   [INPUT] if set, ignores whether any default (except TOPDIR)
                has already been set and resets it anyway
                * to reset TOPDIR, just do it on the command line, as
                  !TOPDIR='/path/to/PINTofALE'
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

history
        Vinay Kashyap (2015aug)
        now if input is scalar, returns scalar (VK; 2015sep)
```

B.78 GMASCL [/external]

```
NAME:
      GMASCL

PURPOSE:

      This is a utility routine to perform basic gray-level pixel
      transformations of images. I think of it as BYTSCL on steroids.
      It is similar to IMADJUST in _Digital Image Processing with MATLAB_
      by Gonzales, Wood, and Eddins.

AUTHOR:

      FANNING SOFTWARE CONSULTING
      David Fanning, Ph.D.
      1645 Sheely Drive
      Fort Collins, CO 80526 USA
      Phone: 970-221-0438
      E-mail: davidf@dfanning.com
      Coyote's Guide to IDL Programming: http://www.dfanning.com

CATEGORY:

      Utilities

CALLING SEQUENCE:

      scaledImage = GMASCL(image)

ARGUMENTS:

      image:         The image to be scaled. Written for 2D images, but arrays
                     of any size are treated alike.

KEYWORDS:

      GAMMA:         The exponent in a power-law transformation (image^gamma). A gamma
                     value of 1 results in a linear distribution of values between
                     OMIN and OMAX. Gamma values less than 1 compress the
                     lower range of values and extend the upper range. Gamma values
                     greater than 1 compress the upper range of values and extend the
                     lower range. The gamma value is constrained to be greater than 1.0

      MAX:           Any value in the input image greater than this value is
```

```
                     set to this value before scaling.

      MIN:           Any value in the input image less than this value is
                     set to this value before scaling.

      NEGATIVE:      If set, the "negative" of the result is returned.

      OMAX:          The output image is scaled between OMIN and OMAX. The
                     default value is 255.

      OMIN:          The output image is scaled between OMIN and OMAX. The
                     default value is 0.
RETURN VALUE:

      scaledImage:   The output, scaled into the range OMIN to OMAX. A byte array.

COMMON BLOCKS:
      None.

EXAMPLES:

      LoadCT, 0                                                 ; Gray-scale colors.
      image = LoadData(11)                                      ; Load image.
      TV, GmaScl(image, Min=30, Max=100)                        ; Similar to BytScl.
      TV, GmaScl(image, /Negative)                              ; Produce negative image.
      power = Shift(ALog(Abs(FFT(image,-1))), 124, 124)         ; Create power spectrum.
      TV, GmaScl(power, Gamma=2.5)                              ; View power specturm with gamma corre
      TV, GmaScl(power, Gamma=2.5, /Negative)                   ; Reverse power spectrum.

RESTRICTIONS:

    Requires SCALE_VECTOR from the Coyote Library:

       http://www.dfanning.com/programs/scale_vector.pro

MODIFICATION HISTORY:

      Written by:  David W. Fanning, 17 February 2006.
```

B.79 HASTOGRAM [/util]

```
function        hastogram
        returns a frequency histogram over a specified grid,
        calculated in a fast and clever way.  really.
        works in haste, without waste.

        works best when the grid is sparsely populated.
        if there are more items than bins, use IDL's
        built-in HISTOGRAM() function instead.

syntax
        f=hastogram(list,x,wts=wts,verbose=verbose)

parameters
        list    [INPUT; required] list of numbers to bin
        x       [I/O] the required binning scheme
                * if not set, assumes a linear grid of size 100 from
                  min(LIST) to max(LIST)
                * if scalar or 1-element vector, assumes this to be the
                  number of bins
                  * if -ve, log grid, else linear
                * if vector with more than 1 element, assumes it to be
                  the bin boundaries.
                * WARNING: if input as scalar, will overwrite with the
                  adopted grip on output

keywords
        wts     [INPUT] if given, appropriately weights each element of LIST
                * default is unity
                * if size does not match size(LIST), will be appropriately
                  interpolated
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

description
        the problem is the following: if the number of bins is large,
        accumulating a frequency histogram by linear search takes too
        long, esp. in IDL if one uses for-loops.
        so, first create a monster array containing both the list elements
        and the grid values.  then sort this array.  this results in an
        array where the list elements are all nicely placed within the
        correct bins.  now, if we've been keeping track of the positions
        of the list elements, it's an easy job to count up the number in
        each bin of the grid.  to do the latter, we first create a new
```

```
        array made up of -1s in positions of list elements and position
        indices for grid values and reorder according to the above sort.
        then, replace each -1 by the nearest non-(-1) from the left.  now
        each list element is assigned the correct bin number.  voila!

restrictions
        X must be sorted in increasing order.
        (no longer, as of Jun02)

subroutines
        KILROY

history
        vinay kashyap (Jan98)
        added kludge to speed up in case max(X) < max(LIST) (VK; Sep98)
        added quit in case X is not sorted in ascending order (VK; JanMMI)
        added keyword VERBOSE, and removed restriction on order of X
          (VK; Jun02)
        changed ints to longs (VK; Aug08)
        converted bracket notation to IDL5+ and fixed bug with inputs being
          array of size [1,N] (VK;Mar10)
```

B.80 HEXED [/util]

```
function        hexed
        converts numbers in the decimal system to hexadecimal strings

syntax
        hxstr=hexed(r,g,b,rescale=rescale)

parameters
        rr      [INPUT; required] (rgb) value to convert to hex
                * can be a scalar or a triple or 2D array of the form [1,3]
                * if input is byte, assumed to go from 0-255
                * if input is integer or float, rescaled according to RESCALE
                * if triple, assumed to be in sequence [r,g,b]
        gg      [INPUT] (green) value to convert to hex
                * used only if rr is a scalar
        bb      [INPUT] (blue) value to convert to hex
                * used only if rr is a scalar

keywords
        rescale [INPUT] if set, and input is not byte, first divides the inputs by
                this number and then multiplies by 255 and then converts to byte
                * if input is float or double, default is 1.0
                * if input is integer, default is 255
        _extra  [JUNK] here only to prevent crashing the program

example
        help,hexed(1,0,0.5,rescale=1)
        #FF007F
        print,hexed([1,0,0.5])
        #FF007F
        print,hexed(transpose([[255,0,0],[0,255,0]]))
        #FF0000 #00FF00

history
        Vinay Kashyap (2015oct)
        cleaned up; allow multiple values input as (N,3) array (VK; 2015nov)
```

B.81 HINODE_XRT_EMIS [/ssw]

```
function        hinode_xrt_emis
        compute and return the combined emissivities of Hinode/XRT
        filters in an array of form [Ntemp,Nfilt] in units of
        [1d-23 ergs cm^5/s] for each pixel

syntax
        xrtemis=hinode_xrt_emis(filter,tgrid,ldbdir=ldbdir,$
        cdbdir=cdbdir,ioneqf=ioneqf,abund=abund,cieroot=cieroot,$
        n_e=n_e,NH=NH,xrteff=xrteff,lstr=lstr,cstr=cstr,$
        /toel,/toph,/toDN,EM0=EM0,pixsize=pixsize,verbose=verbose,$
        chidir=chidir,fH2=fH2,He1=He1,HeII=HeII)

parameters
        filter  [INPUT; required] scalar or array of filter names
                * for Hinode/XRT, acceptable filter names are
                  Al-mesh, Al-poly, C-poly, Ti-poly, Be-thin,
                  Be-med, Al-med, Al-thick, Be-thick,
                  Al-poly/Al-mesh, Al-poly/Ti-poly, Al-poly/Al-thick,
                  Al-poly/Be-thick, C-poly/Ti-poly
                * set to 'all' to get everything
        tgrid   [I/O] the log(T[K]) grid over which the output
                is to be defined
                * if not specified on input, will be set to be
                  the same as that in the emissivity tables,
                  the PoA default, findgen(81)*0.05+4

keywords
        ldbdir  [INPUT; '$CHIANTI'] line emissivity database directory
        cdbdir  [INPUT; '$CONT'] continuum emissivity database directory
        ioneqf  [INPUT; 'ioneq/mazzotta_etal.ioneq'] location of the
                ion fraction tables
        abund   [INPUT; getabund('Grevesse et al')] abundances
        cieroot [INPUT; 'cie'] root name for the files in $CONT
        n_e     [INPUT; 1e9] electron number density in the plasma
        NH      [INPUT; 0] H column density to apply
        xrteff  [I/O] the XRT effective areas; input can be any of the
                following:-
                - a structure that contains the following fields --
                  {TYPE, CHANNEL_NAME, WAVE, EFF_AREA, LENGTH}
                  which can be generated using the SSW routine
                  xrteff = MAKE_XRT_WAVE_RESP(contam_time=contam_time)
                - the name of a save file that contains the previously
                  generated structure, named either XRTEFF or EFF
```

```
                - a flag that indicates that this must be calculated
                  in situ by calling MAKE_XRT_WAVE_RESP() (requires SSW)
                  - this can be accomplished by setting /XRTEFF, but
                    that will prevent it from being returned up.  instead,
                    it is better to do something like
                    xrteff=1 & xrtemis=hinode_xrt_emis(...,xrteff=xrteff,...)
                * on output, will always contain the structure that
                  was read in or calculated
        lstr    [OUTPUT] line emissivities structure, as read in
                from RD_LINE()
        cstr    [OUTPUT] continuum emissivities structure, as read in
                from RD_CONT()
        toel    [INPUT; default=0] if set, returns the output in units
                of [el cm^5/s]
        toph    [INPUT; default=0] if set, returns the output in units
                of [ph cm^5/s]
        toDN    [INPUT; default=0] if set, returns the output in units
                of [DN cm^5/pix]
                * NOTE: TODN overrides TOPH overrides TOEL
        EM0     [INPUT; default=1] if set, multiples the output by
                this value to derive [ph/s] or [ph/cm^2/s] depending
                on whether the input has units [cm^-3] or [cm^-5]
        pixsize [INPUT; default=1.0024 arcsec] the size of a pixel on
                the detector
        xrtl    [OUTPUT] the response due solely to lines
        xrtc    [OUTPUT] the response due solely to continuum
                * both XRTL and XRTC are identical in size and units
                  to the primary output, XRTEMIS
        verbose [INPUT; default=0] controls chatter
        _extra  [INPUT ONLY] pass defined keywords to subroutines
                FOLD_IONEQ : CHIDIR
                ISMTAU : FH2, HE1, HEII
                MAKE_XRT_WAVE_RESP : INDEX, CONTAM_THICK, CONTAM_TIME

history
        vinay kashyap (Jun2007)
        added keywords TOEL, PIXSIZE, XRTL, XRTC (VK; Mar2008)
        changed call from CALC_XRT_EFFAREA() to MAKE_XRT_WAVE_RESP() (VK; Apr2009)
        bugfix: FILTER was failing to match to XRTEFF.NAME when specified
          (VK; Nov2009)
```

B.82 HIPD_INTERVAL [/stat]

```
function        hipd_interval
        computes and returns the interval [lower_bound,upper_bound] at a
        specified confidence level that includes the highest probability
        densities.  by definition, this is the shortest possible interval.

syntax
        hpd=hipd_interval(f,x,/fsample,clev=clev,pdfnorm=pdfnorm,$
        fmode=fmode,verbose=verbose)

parameters
        f       [INPUT; required] the array for which the confidence interval
                must be computed
                * assumed to be a density function unless FSAMPLE is set
        x       [INPUT; optional] abscissae values
                * if not given, and F is a density function, then taken
                  to be the array indices
                * ignored if FSAMPLE is set
                * if N(X).GT.1 but N(X).NE.N(F), X is ignored and
                  FSAMPLE is set automatically

keywords
        fsample [INPUT] if set, assumes that F is a set of samples from a
                density function, as opposed to being the density function
                itself
        clev    [INPUT] confidence level at which to compute the intervals
                * default is 0.68
                * if < 0, abs(CLEV) is used
                * if > 1 and < 100, then assumed to be given as a percentage
                * if > 100, then 1-1/CLEV is used
        pdfnorm [INPUT] if set, forces F to integrate to abs(PDFNORM)
                * if explicitly set to 0, does not normalize F at all
                * if not set, normalizes to 1
                * ignored if FSAMPLE is set
                * WARNING: do not use this keyword unless you know
                  what you are doing
        fmode   [OUTPUT] the mode of the distribution
        verbose [INPUT] controls chatter
        _extra  [INPUT ONLY] pass defined keywords to subroutines
                MODALPOINT: EPS

subroutines
        MODALPOINT
```

```
description
        * if density function, find the cumulative integral around the
          mode and interpolate at CLEV to derive the HPD interval
        * if array of sample values, find all intervals corresponding to
          CLEV that contain the mode and pick the smallest of the lot.
          this is a method devised by Vinay K and Taeyoung Park during
          the course of developing BEHR.
        * works well only for unimodal distributions (and for multimodal
          distributions where all the modes are within the interval),
          but that's better than nothing.
        * note that this interval is not invariant under transformations.
          for that, one must use equal-tail intervals, see eqt_interval()

example
        for i=1,20 do print,hipd_interval(randomn(seed,10000L)*i,/fsample)

history
        vinay kashyap (Mar2006)
        bug correction with FSAMPLE cdf (VK; Apr2006)
        added keyword FMODE (VK; Nov2006)
        bug correction with F(X) case (VK; Apr2007)
        now handles NaNs in input (VK; Apr2014)
```

B.83 HIRES2LOWRES [/esempio]

```
procedure     hires2lowres
        given filenames containing HETG HEG and MEG PHA data,
        calculated ARFs, and ACIS ARFs and RMFs, predict what would
        be observed with ACIS-S or ACIS-I (without gratings)

syntax
        make_lowres,hphafile,mphafile,hgarffile,mgarffile,$
        hhetgwvl,hhetgcts,mhetgwvl,mhetgcts,hetgwvl,hhetgflx,$
        hhetgflxerr,mhetgflx,mhetgflxerr,tothetgflx,tothetgflxerr,$
        aciswvl,aciscts,aciserr,hphahdr,mphahdr,hgarfhdr,mgarfhdr,$
        acisarffile,acisrmffile,debug=debug

parameters
        hphafile     [INPUT; required] PHA type I spectral file
                     containing the HEG HETG spectrum for a given
                     order
        mphafile     [INPUT; required] PHA type I spectral file
                     containing the MEG HETG spectrum for a given
                     order
        hgarffile    [INPUT; required] Gratings ARF corresponding to
                     the HEG PHA file
        mgarffile    [INPUT; required] Gratings ARF corresponding to
                     the MEG PHA file
        hhetgwvl     [OUTPUT; required] The midpoints of the HEG
                     wavelength bins
                     * units are Angstroms
        hhetgcts     [OUTPUT; required] The background-subtracted
                     counts spectrum extracted from the HEG PHA file
                     * units are [cts/s/AA]
        mhetgwvl     [OUTPUT; required] The midpoints of the MEG
                     wavelength bins
                     * units are Angstroms
        mhetgcts     [OUTPUT; required] The background-subtracted
                     counts spectrum extracted from the MEG PHA file
                     * units are [cts/s/AA]
        hetgwvl      [OUTPUT; required] The midpoints of the common
                     HEG, MEG, and HEG+MEG flux spectrum's
                     wavelength grid
                     * grid extends ~0-30 AA at HEG resolution
        hhetgflx     [OUTPUT; required] Adaptively-smoothed,
                     background-subtracted HEG flux spectrum
                     * units are [ph/s/cm^2/AA]
        hhetgflxerr  [OUTPUT; required] The error on the HEG flux
```

```
                spectrum
        mhetgflx     [OUTPUT; required] Adaptively-smoothed,
                     background-subtracted MEG flux spectrum
                     * units are [ph/s/cm^2/AA]
        mhetgflxerr  [OUTPUT; required] The error on the MEG flux
                     spectrum
        tothetgflx   [OUTPUT; required] Adaptively-smoothed,
                     background-subtracted coadded HEG+MEG flux
                     spectrum
                     * units are [ph/s/cm^2/AA]
        tothetgflxerr [OUTPUT; required] The error on the coadded
                     HEG+MEG flux spectrum
        aciswvl      [OUTPUT; required] The midpoints of the
                     predicted ACIS spectrum's wavelength grid
        aciscts      [OUTPUT; required] The predicted ACIS spectrum
                     * units are [cts/s/AA]
        aciserr      [OUTPUT; required] The error on the predicted
                     ACIS spectrum
        hphahdr      [OUTPUT; required] The header of the HEG PHA
                     file
        mphahdr      [OUTPUT; required] The header of the MEG PHA
                     file
        hgarfhdr     [OUTPUT; required] The header of the HEG gARF
                     file
        mgarfhdr     [OUTPUT; required] The header of the MEG gARF
                     file
        acisarffile  [INPUT; required] The ACIS ARF
        acisrmffile  [INPUT; required] The ACIS RMF

keywords
        debug        [INPUT] controls debugging chatter

restrictions
        requires conv_rmf, smoothie, and rebinw from PINTofALE,
        available at http://hea-www.harvard.edu/PINTofALE/

history
        Owen Westbrook, Peter Mendygral, and John Slavin (Mar07)
```

B.84 HRS2DEG [/util]

```
function        hrs2deg
        converts sexagesimal angles to decimal degrees and returns a
        numerical array of decimal angles

parameters
        hms     [INPUT; required] string array of sexagesimal angles
                * may even be a string of FSEP separated sexagesimals

keywords
        fsep    [INPUT] field separator in HMS
                * if FSEP=":", only the first 3 are used
        dec     [INPUT] if set, does not multiply by 15 (the RA v/s Dec thing)
                * if the H part is -ve, automatically set
        _extra  [INPUT] allows passing defined keywords to
                STR2ARR: SEP, SQUISH

subroutines
        STR_2_ARR
        see also DEG2HRS

history
        vinay kashyap (Nov98)
        now calls STR_2_ARR instead of STR2ARR (VK; Mar08)
```

B.85 HULLSORT [/util]

```
function        hullsort
        reorder a 2D list from outside in and return the indices of
        the reorderd list.  At each stage, compute the convex hull
        and remove the "farthest" point from the hull centroid.

syntax
        iout=hullsort(xin,yin,xout=xout,yout=yout,xcen=xcen,ycen=ycen,$
        areap=areap,/bycen,twait=twait,verbose=verbose)

parameters
        xin     [INPUT; required] 1D array of x-coordinate points
        yin     [INPUT; required] 1D array of y-coordinate points
                * sizes must match
                * only useful if there are more than 4 points

keywords
        xout    [OUTPUT] reorderd XIN
        yout    [OUTPUT] reorderd YIN
        xcen    [OUTPUT] centroid x-coordinate computed at each step
        ycen    [OUTPUT] centroid y-coordinate computed at each step
        areap   [OUTPUT] area covered by the convex hull at each stage
        bycen   [INPUT] if set, eliminates points from convex hull based
                on distance to center
                * default is to eliminate those farthest from all other
                  points on the convex hull
        twait   [INPUT] waiting time for replotting on screen [seconds]
                * only applies if VERBOSE>10
                * default is 0.001
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

example
        .run hullsort

subroutines
        QHULL (IDL built-in)
        KILROY
        AREAPOLY

history
        vinay kashyap (MMXII.VII)
        added call to AREAPOLY() (VK; MMXII.X)
```

B.86 ID2EMIS2ID [/]

```
function        id2emis2id
        takes an ID structure, strips it into component parts,
        reads in new emissivities, and puts them back in.

syntax
        newidstr=id2emis2id(idstr,ldbdir,dWVL=dWVL,verbose=verbose,$
        eps=eps,/incieq,mapping=mapping,pres=pres,logP=logP,n_e=n_e,$
        chifil=chifil,chidir=chidir,eqfile=eqfile)

warning
        new emissivities will not necessarily reflect the fluxes
        as distributed among the components.  also, if the database
        is changed, there is no reason to believe a priori that
        the correct lines are really read back in.  we strongly
        recommend using a high (say 10) verbosity level.

parameters
        idstr   [INPUT; required] ID structure, see LINEID for description
        ldbdir  [INPUT] directory in which to look for line database
                * default is "$CHIANTI"
                * if array size matches the number of features, is
                  distributed among components appropriately; if size
                  matches that of total number of lines, maps one-to-one
                  onto line list; if size is incompatible with IDSTR,
                  then uses only what is in the first element.

keywords
        dWVL    [INPUT] slack in wavelength search in which to find the
                matching line (this comes in handy when databases are
                being changed around)
                * default is 0.005
        verbose [INPUT] controls chatter
        _extra  [INPUT ONLY] pass defined keywords to
                RD_LIST: EPS,INCIEQ,MAPPING,PRES,LOGP,N_E,CHIFIL,CHIDIR,EQFILE

restrictions
        requires subroutines:
          RD_LIST, RD_LINE, FOLD_IONEQ, RD_IONEQ, READ_IONEQ,
          SYMB2ZION, ZION2SYMB, LAT2ARAB, CAT_LN, LINEFLX,
          INICON, WHEE, GETABUND, RDABUND, SYZE
        requires IDL 5.3+

history
```

```
vinay kashyap (AugMM)
catch if correct emissivities not found; added keywords
  VERBOSE, DWVL; allowed LDBDIR to be array (VK; JanMMI)
```

B.87 ID_TO_FITPAR [/fitting]

```
procedure       id_to_fitpar
        given the ID structure resulting from LINEID, figures out
        initial fitting parameters for use in, e.g., FITLINES, FIT_LEVMAR,
        etc.

syntax
        id_to_fitpar,idstr,pars,ties,thaw,pos=pos,wdt=wdt,flx=flx,$
        lsfwdt=lsfwdt,shackle=shackle,leeway=leeway,perr=perr,werr=werr,$
        ferr=ferr

parameters
        idstr   [INPUT; required] ID structure containing the observed
                wavelengths and appropriate matches
        pars    [OUTPUT] initial values of all the parameters set up in
                a single array for consumption by, e.g., FIT_LEVMAR or
                X3MODEL.
        ties    [OUTPUT] constraints on the parameters
        thaw    [OUTPUT] integer array of same size as PARS, signaling
                frozen (0) or thawed (1) parameter
                * all parameters that are SHACKLEd are frozen

keywords
        pos     [OUTPUT] initial-guess positions of all lines
        wdt     [OUTPUT] initial-guess widths of all lines
        flx     [OUTPUT] initial-guess fluxes of all lines
       epithet [OUTPUT] species label (e.g. 'OVII' or 'FeXIX') string array
        lsfwdt  [INPUT] width of the instrument's line spread function, and
                the minimum value of WDT
                * if not defined, assumed to be same as PERR
                  (on the theory that your position determination is uncertain
                  on the same order as the LSF width)
        shackle [INPUT] specify how to handle multiple ID components:-
                "position" => line centers are fixed relative to location
                  of line of maximum strength
                "width" => freeze all WDT at maximum determined value OR
                  value of WERR if given
                "flux" => relative strengths of fluxes are held fixed
                * if array, only the first element is considered
                * if integer (as in /shackle), all 3 of above are set
                * any two (or all 3) may be specified simply as "flux & pos",
                  "width and flux", "position,width,flux", etc.
                * all affected parameters are frozen
        leeway  [INPUT] specify how to handle multiple ID components:-
```

```
                "position" => relative locations of line centers are held
                  to within a narrow range of position of line of maximum
                  strength, defined by PERR
                "width" => all WDT are constrained to be > minimum determined
                  width, and < maximum determined width, and this range may
                  be further expanded by WERR
                "flux" => relative strengths of fluxes are held to within
                  a narrow range defined by FERR
                * obviously, SHACKLE takes precedence if both are set
                * if array, only the first element is used
                * if integer (e.g., /leeway), all of above are set
                * unlike SHACKLE, all affected parameters are kept THAWed.
        perr    [INPUT; default=1.0] slippage allowed in POS
                * setting this is crucial in case the IDs include lines
                  from higher orders or unknown IDs, because this will
                  tell the program that deviations larger than PERR from
                  the location of the observed line are suspect.
                * if LSFWDT is not set, PERR constitutes a >lower limit< to WDT
        werr    [INPUT] allowed excess variation in range of WDT
                * default is 0.5*PERR[0]
                * subtracted from min and added to max if LEEWAY requires it
        ferr    [INPUT; default=0.1] allowed fractional error in FLX
                * has an effect only if LEEWAY is set
                * NOTE: for PERR,WERR,FERR, if size does not match that of
                  POS,WDT,FLX respectively OR the number of components in
                  IDSTR, only the first element is used
       numer   [INPUT; default = 0] if set, ties specified via leeway, perr,
                 werr, and ferr will be specified numerically rather than
                 symbolically
       nozero  [INPUT; default = 0] if set, then fluxes are constained to
                 be positive. if set negative, then fluxes are constrained
                 to be negative
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (Nov98)
        added keyword LSFWDT (VK; Mar99)
        changed call to INITSTUFF to INICON (VK; 99May)
       added EPITHET keyword (LL; Feb04)
       added NUMER keyword (LL; Jul05)
       BUGFIX: for multiplet component leeway width specification (LL; Jul 05)
       had used lower bound for position instead of width
       added NOZERO keyword (LL; Jul05)
        BUGFIX: for multiplet component leeway position specification
          was written with the wrong sign (VK; May06)
```

B.88 IDL_RELEASE [/external]

```
Project     : SOHO - CDS

Name        : IDL_RELEASE

Purpose     : check if IDL release version within specified range

Category    : system

Explanation :

Syntax      : IDL> a=idl_release(lower=lower,upper=upper)

Examples    :

Inputs      : None

Opt. Inputs :

Outputs     : 1/0 if IDL version is within specified range

Opt. Outputs: None

Keywords    : LOWER = lower version to check
              UPPER = upper version to check
              INCLUSIVE = make check inclusive
              VERS = IDL version

Common      : None

Restrictions: None

Side effects: None.

History     : Version 1,  27-Feb-1997,  D M Zarro.  Written

Contact     : DZARRO@SOLAR.STANFORD.EDU
```

B.89 IDLABEL [/]

```
function        idlabel
        return a string array containg a nicely formatted set of labels,
        appropriate for each of the IDs in an ID structure.

syntax
        label=idlabel(idstr,idx,/Zbeda,/Ibeda,/Wbeda,/Lbeku,$
        wform=wform,wstyle=wstyle,sep=sep,intex=intex, ziform=ziform)

parameters
        idstr   [INPUT; required] the ID structure containing the features
                with attached IDs. (see LINEID for description)
        idx     [OUTPUT] an integer array pointing out which label belongs
                to which feature.  this is of use in case of multiple IDs.
                * begins from 0, not 1

keywords
        Zbeda   [INPUT] if set, does not include the atomic symbol in label
        Ibeda   [INPUT] if set, does not include the ionic state in label
        Wbeda   [INPUT] if set, does not include the line wavelength in label
        Lbeku   [INPUT] if set, includes a description of the transition,
                in the form of the level-designation and electronic
                configuration (if available) in label
        wform   [INPUT] wavelength format, relevant if Wbeda=0
                * default: ’f7.2’
                  if WSTYLE = 1, unadorned WFORM
                  if WSTYLE = 2, automatically prepends $\lambda$:
                        ’"!4k!3 ",’+WFORM
                  if WSTYLE = 4, automatically appends \AA:
                        WFORM+’,"’+string(byte(197))+’"’
                  if WSTYLE = 8, automatically appends keV: WFORM+’," keV"’
        wstyle  [INPUT] bit-style flag shortcut to various WFORMs
                * see above for effect on WFORM
                * if -ve, uses the wavelength of the ID’d feature, not
                  the wavelength of the ID itself
        sep     [INPUT] separator between "Z Ion", wvl, and description
                * default is "<tab>", unless INTEX is set, in which case
                  the default is " & "
        intex   [INPUT] if set, encloses the description (see LBEKU) within
                "$...$"
        _extra  [INPUT] pass defined keywords to subroutines
                ZION2SYMB: ZIFORM

restrictions
```

```
        requires input to be in format generated by LINEID, CAT_ID, etc.
        requires subroutines
                ZION2SYMB [INICON]

examples
        l=idlabel(idstr,idx,wform='f5.1')
        l=idlabel(idstr,/Wbeda)
        l=idlabel(idstr,wstyle=4)
        l=idlabel(idstr,wstyle=6)
history
        vinay kashyap (AugMM)
        added keywords SEP,INTEX; implemented LBEKU (VK; JanMMI)
        back-compatibility: STRMID must have 3 args (VK; FebMMI/A.Maggio)
        bug fix: IDNAM[2] is ID1, not ID2 (VK; MayMMVI/L.Lin)
```

B.90 INICON [/util]

```
procedure       inicon
        return atomic symbols, names, ionization potentials, fundamental
        and other fun constants, atomic weights, roman numerals

syntax
        inicon,atom=atom,aname=aname,amu=amu,fip=fip,fundae=fundae,$
        roman=roman,/help

parameters      NONE

keywords
        atom    [OUTPUT] atomic symbols
        aname   [OUTPUT] names of elements
        amu     [OUTPUT] structure containing atomic weights in AMU (1H=1)
                * AMU.(Z) is an array, with first element the atomic weight
                  for terrestrial composition, then the AMUs of the isotopes
                  in order of abundance (i.e., most abundant is 2nd element,
                  second most abundant is 3rd element, etc. -- but not
                  completely implemented)
        fip     [OUTPUT] first ionization potentials [eV]
                * set to -1 if not known
        fundae  [OUTPUT] structure containing some useful physical and
                astronomical constants
                * just do help,fundae,/str -- should be obvious
        roman   [OUTPUT] roman numerals, upto max(ATOM)+1
                * beware that CHIANTI has a function named ROMAN()
                  which returns numerals that are offset by 1 compared
                  to this variable.

                NOTE: in IDL versions prior to 5, the keywords must be
                pre-defined if output is desired

requires
        CREATE_STRUCT

history
        modified from INITSTUFF (VK; 99May)
        bug correction -- ROMAN wasn't working (VK; 99Jul)
        added Jupiter (VK; MIM.XI)
        added keV-to-Ang (VK; MMII.I)
        added planetary data (VK; MMII.IX)
        changed AMU tag names from symbol to full names (VK; MMII.X)
        corrected AMU value (JD; IMVIM.I)
```

```
added JouleV, ergeV, and arcsr (VK; IMVIM.V)
corrected Bode's Law and extended to 12 "planets" (VK; MMV.VII)
changed Planets field to "Classical Planets", moved Pluto to
  separate Plutoid field (VK; MMVIII.VII)
added Thompson cross section (per JJD req), Compton wvl
  (VK; MMXI.XI)
new IAU definition for A.U. (VK; MMXII.IX)
added new Pluto and Charon measurements from New Horizons (VK; MMXV.VII)
```

B.91 INITALE [/scrypt]

```
script  initale
        extremely flexible initialization script for the Package for
        Interactive Analysis of Line Emission (PINTofALE); defines some
        useful variables as system variables.  Examples and scripts in
        the standard distribution use these variables.

        NOTE: Strictly speaking, none of these system variables are
        _necessary_ in order to use basic PINTofALE functionality.
        However, they set up the default values for some commonly
        used keywords and usually must be explicitly included in
        calls to PINTofALE routines.  Most scripts and documentation
        examples do use these directly, and some routines set their
        default values at run-time if the system variables are defined.
        An additional use of this script is that it sets up the !PATH
        to include PINTofALE, CHIANTI, and IDL-Astro if they are not
        already included.

syntax
        .run initale
        (or)
        .run /full/path/initale

system variables and hardcoded default values
        !PoA     current version number
        !TOPDIR  the top-level PINTofALE directory
                 * first checks for environment variables
                        PINTofALE, PoA, SCARDIR, and SCAR
                   in that order.  if none of them exist, then uses
                   value deduced from the location of _this_ script.
        !LDBDIR  [!TOPDIR+'emissivity/chianti'] directory of line emissivities
        !CDBDIR  [!TOPDIR+'emissivity/cont'] directory of continuum emissivities
        !CHIDIR  [!TOPDIR+'CHIANTI'] path to CHIANTI installation
                 NOTE: !TOPDIR is added to initial values of above 3 ONLY IF:
                 -- !TOPDIR is not already contained in them
                 -- the first character is not a "$"
                 -- the first character is not a "/"
                 -- for CHIANTIv4+, path name must include the "dbase" part
        !ATOMDB  [!TOPDIR+'atomdb/'] directory of local installation of ATOMDB
        !APECDIR where the IDL tools of APED are
                 * first checks for environment variable APEC_DIR, and if
                   that is missing, looks in succession in
                   -- !ATOMDB+'apec_v11_idl/'
                   -- !TOPDIR+'apec_v11_idl/'
```

```
                   -- !TOPDIR+’atomdb/apec_v11_idl/’
        !ARDB    [!TOPDIR+’ardb/’] the Analysis Reference DataBase directory,
                 contains sundry useful documents
        !CEROOT  [’cie’] root prefix for continuum emissivity files
        !IONEQF  [’ioneq/bryans_etal_09.ioneq’] path name relative to
                   !CHIDIR of the ion-balance file to be used
        !ABREF   [’grevesse et al.’] abundance reference
        !CALDB   [’/data/caldb/’] directory containing instrument
                    calibration data products
        !METALS  [0.0] [Fe/H] relative to !ABREF
        !ABUND   abundances of elements H..Zn corresponding to !ABREF and
                 !METALS
        !GASPR   [1e15 cm^-3 K] gas pressure
        !LOGPR   [15] log10(gas pressure) -- if set, overrides !GASPR
        !EDENS   [1e10 cm^-3] electron density -- if set, overrides use of
                  !GASPR and !LOGPR
        !LOGT    [findgen(81)*0.05+4] the default temperature grid
        !DEM     [dblarr(81)+1d12] a sample DEM(!LOGT)
        !NH      [1e18 cm^-2] H column density
        !FH2     [0.26] fraction of molecular H2 relative to HI
        !HE1     [1e17 cm^-2] He I column
        !HEII    [1e16 cm^-2] He II column
        !WMIN    [1.239854 Ang] minimum wavelength
        !WMAX    [3000.0 Ang] maximum wavelength
        !VERBOSE [5] verbosity -- controls chatter
        !ATOM    atomic symbols
        !AMU     atomic weights
        !FUNDAE  a whole bunch of fundamental constants
        !FIP     firt ionization potentials [eV]
        !ROMAN   roman numerals, upto max(!ATOM)+1
        !AA      [’!3’+string(byte(197))+’!X’] the symbol for Angstrom
        !ANGSTROM       An array of Angstrom symbols, should work for
                        every conceivable font choice.

control variables
        factory  determines whether to reset all the system variables to
                 the "factory default" or not.  set this variable to 0 or 1
                 prior to running the script.
                 * if explicitly set to 0, then only initializes those
                   variables which have not been defined yet
                 * if PARFIL is defined, automatically gets unset
                 * if set, or is not defined, then overwrites all variables
                   with hardcoded defaults
                 -- one may want to use FACTORY=0 when some of the variables
                    have been set (e.g, !TOPDIR) during the session without
```

```
                    an initial call to this script, or when a limited number
                    of the variables are being reset via a parameter file.
                 -- upon execution, if FACTORY is not set, it will be set to 0
        parfil   if set to a named parameter file that contains new definitions
                 of the defined system variables.  They must be placed one on
                 each line, and must be fully legal IDL statements.
                * if not set, will automatically look in
                  !ARDB+'initale.par'
                * DO NOT try to define the following in PARFIL:
                   !ABUND, !ATOM, !AMU, !FUNDAE, !FIP, !ROMAN, !PoA
        yCHIANTI if set then checks to see whether the CHIANTI distribution is
                 available for use and includes it if it is not.
        yATOMDB  if set then checks to see whether the ATOMDB IDL routines
                 are accesible and adds them if they are

calls subroutines
        GETABUND
        INICON
        PEASECOLR

restrictions
        subroutine WHICH works only in UNIX, but it is used only if none
          of the environment variables PINTofALE, PoA, SCARDIR, or SCAR
          are set.
        Unlike normal IDL variables, system variables, once defined,
          cannot change their size, though their values may be changed.
          Thus, care must be taken to match the expected sizes of the
          following variables to the hardcoded defaults if they are to
          be redefined within PARFIL:
          !ABUND, !LOGT, !DEM, !ATOM, !AMU, !FUNDAE, !FIP, !ROMAN

history
        Vinay Kashyap(OctMM)
        set FACTORY to 0 if undefined; changed !ABREF default; added !ARDB;
          !TOPDIR now automagic (VK; DecMM)
        added !PoA, !CALDB, !LOGT, !DEM, and !AA (VK; JanMMI)
        fixed bug with !TOPDIR becoming an array if none of the env vars
          were defined; change prompt for large !VERBOSE (VK FebMMI/Dan Dewey)
        changed call to WHICH to WHEREIS (VK; Dec2001)
        commented out call to WHEREIS by using the output of HELP, and now
          adds !TOPDIR/pro to !PATH if it doesn't exist; also check for
          other packages such as IDL-ASTRO and CHIANTI; more robust to
          OS filename dependencies for v5.3+ due to extensive use of
          FILEPATH function; added !ATOMDB, !APECDIR, and yATOMDB
          (VK; Jun2002)
```

```
added call to PEASECOLR (VK; Jul02)
some people have IDL-ASTRO installed under 'astrolib' (VK; Aug02)
added !ANGSTROM; made !AA robust to font choices (VK; Dec'02)
suppose environment variable is defined, but incorrectly?  i.e.,
  what if $PINTofALE, $SCARDIR, etc. do not exist? (VK; May'03)
if IDL_DIR is not defined, assumes that IDL library is in anything
  that has '...idl/lib' (VK; Jul'03)
corrected case where if env variable defining TOPDIR had a trailing
  "/", would miss matching it in !PATH and not reset the order of
  directories (VK; MarMMIV)
made a few of the system variables read-only (VK; May04)
made windows compatible, by and large (VK; Apr05)
forced IDL-Astro to move ahead of Chianti's astron directory
  in !PATH (VK; FebMMVII)
removed early call to strsplit() and forced restoration of
  correctly compiled versions of STRSPLIT, UNIQ, and STR_2_ARR
  (VK; JunMMVII)
bugfix: wasn't setting TOPDIR correctly when it had to be determined
  from the location of this file (VK; OctMMIX)
changed default for !IONEQF from Mazzotta et al. to Bryans et al. (2009)
  (VK; MayMMXIII)
```

B.92 INTERSECT [/util]

```
procedure       intersect
        procedure to obtain common elements of two numeric arrays
        sorted in ascending order and with no duplicates

syntax
        intersect,lis1,lis2,y1y2,y1n2,n1y2,/val,$
        ny1y2=ny1y2,ny1n2=ny1n2,nn1y2=nn1y2,verbose=verbose

parameters
        lis1    [INPUT; required] input list #1, array
        lis2    [INPUT; required] input list #2, array
        y1y2    [OUTPUT; required] elements in LIS1 also in LIS2
        y1n2    [OUTPUT] elements in LIS1 that are not in LIS2
        n1y2    [OUTPUT] elements not in LIS1 that are in LIS2

keywords
        val     [INPUT] if set, returns the actual values in x1x2 rather
                than the array position numbers
        ny1y2   [OUTPUT] number of elements in Y1Y2
        ny1n2   [OUTPUT] number of elements in Y1N2
        nn1y2   [OUTPUT] number of elements in N1Y2
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

description
        Y1Y2 contains indices of elements of LIS1 that are common to both LIS1 and LIS2.
        Y1N2 contains indices of elements of LIS1 that are not in LIS2.
        N1Y2 contains indices of elements of LIS2 that are not in LIS1.
        if Y1Y2/Y1N2/N1Y2 is empty, it is set equal to -1L
        if keyword VAL is set, actual array elements are returned instead of indices,
        and if Y1Y2/Y1N2/N1Y2 is empty, it is set equal to !values.F_NAN

history
        algorithm by mala viswanath
        idl version (but not optimized for it) by vinay kashyap (8/27/1992)
        added keyword VAL (3/26/93)
        added keyword VERBOSE (4/14/02)
        added keywords NY1Y2,NY1N2,NN1Y2, cleaned up edge case outputs, check for sorted
```

B.93 IONABS [/]

```
function        ionabs
       returns the photoelectric absorption cross-sections for a given
        photon energy for specified chemical composition, and ion fractions.
        Uses ground state photoionisation cross-sections computed using the
        fortran code of Verner, D.A., Ferland, G.J., Korista, K.T., & Yakovlev,
        D.G., 1996, ApJ, 465, 487 and is supplemented with high resolution
        cross-sections in the vicinity of 1s-2p resonances for
        O (from Garcia et al., 2005, ApJS, 158, 68),
        Ne (from Juett et al., 2006, ApJ, 648, 1066), and
        C (from Hasogle et al, 2010, ApJ, 724, 1296).

syntax
        sigabs=ionabs(w,abund=abund,ionfrac=ionfrac,vfkydir=vfkydir,/ikeV,$
        nconsec=nconsec,DEM=DEM,logt=logt,verbose=verbose,noHeH=noHeH,$
        icrstr=icrstr,chidir=chidir,eqfile=eqfile)

parameters
        w       [INPUT; required] photon energy values at which to compute
                the absorption cross-section
                * default units are [Ang], unless IKEV is set, when they are
                  assumed to be [keV]

keywords
        abund   [INPUT] abundances relative to H=1
                * default is to use Grevesse & Sauval
        ionfrac [I/O] the ion fraction for all elements
                * must be a 2D array of size (NZ,NION), where
                  NION=NZ+1 and usually NZ=30
                * note that in normal cases, total(IONFRAC,2)
                  is an array of 1s.
                * if not given, assumed to be all neutral, IONFRAC[*,0]=1
                  - unless DEM is given, see below
                * if non-zero scalar, assumed to be all neutral
                  - DEM is ignored
                * if given as an array of size NZ, each element
                  is assumed to have a total of that fraction,
                  and all the ions have the same fraction
                  - this is clearly an artificial case, to be used mainly
                    for debugging and the like
                  - DEM is studiously ignored
                * if given as a 1D array of size not matching NZ,
                  is ignored with a warning and the default is used
                  - unless DEM is given, see below
```

```
                * if given as a 2D array of size not matching (NZ,NZ+1),
                  program quits with an error
                  - unless DEM is given, see below
                * if DEM (see below) is given, is read in as a 3D array
                  of size (NT,NZ+1,NZ1) from the database (using the PINTofALE
                  function RD_IONEQ(), which calls the CHIANTI procedure
                  READ_IONEQ), weights the fractions according to the DEM
                  and is converted into a 2D array of size (NZ,NZ+1)
        vfkydir [INPUT] where to find the save files that contain the
                cross-sections calculated for each ionization state
                * default is '$ARDB'
                * NOTE: the cross-sections are not read in if the keyword
                  ICRSTR is properly defined
        ikeV    [INPUT] if set, assumes that W are in units of keV
        nconsec [INPUT] number of grid points that define the cross-sections
                that must fall within one bin of the user defined grid before
                the switch from averaging to interpolating occurs
                * default=1
        DEM     [INPUT] the Differential Emission Measure that will be
                used to weight the different temperatures to compute the
                ionization fraction
                * looked at only if IONFRAC is not supplied as input
                * size must match LOGT, otherwise ignored
        logT    [INPUT] the temperature grid over which DEM is defined
                * size must match DEM, otherwise DEM is ignored
        noHeH   [INPUT] if set to any number other than 1 or 2, excludes
                H and He from the cross-sections
                * if set to 1, only excludes H
                * if set to 2, only excludes He
        icrstr  [I/O] the cross-sections calculated for each ionization
                state, stored in a structure of structures of the form
                ICRSTR.(Z) = {EVSPLAC, CROSSI}
                * if given on input, will use these numbers
                * if the structure does not contain enough tags, or if
                  any one of the substructures does not contain enough
                  tags, will be read in using the data in VFKYDIR
        verbose [INPUT] controls chatter
        _extra  [INPUT ONLY] pass defined variables to subroutines
                -- RD_IONEQ : CHIDIR, EQFILE

subroutines
        GETABUND()
        RD_IONEQ()
        READ_IONEQ
        INICON
```

```
        GETPOADEF()

history
       written as IONTAU by Jeremy Drake (Apr07)
       further shoehorned into PoA format by Vinay Kashyap (Apr07)
        bug fixes and corrections and changed behavior of IONFRAC
          (VK; May09)
        updated references (JJD; Aug11)
        BUG: DEM was not being normalized when weighting IONFRAC (JJD; Nov11)
        added call to GETPOADEF (VK; Aug15)
```

B.94 IS_DIR [/external]

B.95 IS_KEYWORD_SET [/util]

```
function        is_keyword_set
        a wrapper to KEYWORD_SET(), in order to produce the
        same behavior with vector [0] as with IDL 5.6 and prior

parameters
        key     [INPUT] the keyword to test for

keywords
        none    isn't this ironic?

description
        prior to IDL 5.6, keyword_set(0) returned false and
        keyword_set([0]) returned true.  this behavior changed
        in IDL 5.6 to match the documentation of keyword_set,
        so that both 0 and [0] now return false.  it is far too
        difficult to change the code logic of all the programs
        written with old IDLs, so better to have a wrapper
        that duplicates the same behavior instead.

history
        Vinay Kashyap (Mar2006)
```

B.96 ISMTAU [/]

```
function        ismtau
        returns the optical depth for a given photon energy at specified
        column density.  i.e., the $\tau$ of $e^{-\tau}$.

        ingredients:
        * in the UV/Opt/IR regime, calls IDLastro's FM_UNRED.PRO
        * in the EUV regime, cannibalizes ISMEUV.PRO for
          -- HI photoionization (Spitzer 19??, "Physical Processes in the
             Interstellar Medium", p105), between 0.03 keV (413.3A) and 911.75A
          -- HeII photoionization (ibid), between 0.03 keV (413.3A) and 911.75A
          -- HeI with auto-ionization resonances (Marr & West 1976;
             Oza 1986, Phys.Rev.A 33 824; Fernley et al. 1987, J.Phys.B 20,
             6457; Rumph, Bowyer, & Vennes 1994, AJ 107, 2108) below 503.97A
          -- NOTE: these are extended down to 43.657 AA when the X-ray
             calculations are not being done.
        * in the X-ray regime (0.03 - 10 keV), either
          -- Morrison & McCammon (1983, ApJ 270, 119) polynomial fit between
             0.03 keV and 10 keV, or
          -- Balucinska-Church & McCammon (1992, ApJ 400, 699) cross-sections,
             which allows abundance variations, or
          -- Wilms, Allen, & McCray (2000, ApJ, in press) calculations
             that use cross-sections from Verner & Yakovlev (1995, A&AS,
             109, 125), which also allows abundance variations
          -- Verner, Ferland, Korista, & Yakovlev (1996, ApJ, 465, 487)
             ground state photoionisation cross-sections that allows for
             abundance variations (down to 911.75 Ang)
        * in the hard X-ray regime and beyond (>10 keV),
          -- Tanaka & Bleeker (1977, ???) power-law extended past 10 keV
        * Molecular H (Cruddace et al. 1977, ApJ 187, 497; Kashyap et
          al. 1992, ApJ 391, 684) between 12.3985A (1 keV) to 911.75A

        to extract only the cross-sections, set NH=1
        to obtain transmission factors, use exp(-ismtau(...))

syntax
        tau=ismtau(w,NH=NH,fH2=fH2,He1=He1,HeII=HeII,Fano=Fano,/ikev,$
        /vion,/wam,/bam,/mam, abund=abund,noHeH=noHeH,verbose=verbose,$
        tauH1=tauH1,tauH2=tauH2,tauHeII=tauHeII,tauHe2=tauHe2,tauHe1=tauHe1,$
        tauX=tauX,tauTB=tauTB,$
        R_V=R_V,LMC2=LMC2,AVGLMC=AVGLMC,ExtCurve=ExtCurve,$
       fmgamma=fmgamma,fmx0=fmx0,fmc1=fmc1,fmc2=fmc2,fmc3=fmc3,fmc4=fmc4,$
        icrstr=icrstr,ionfrac=ionfrac,vfkydir=vfkydir,nconsec=nconsec,$
        DEM=DEM,logT=logT,chidir=chidir,eqfile=eqfile)
```

```
parameters
        w       [INPUT; required] photon energy values at which to compute
                the optical depth.
                * default units are [Ang], unless IKEV is set, when they are
                  assumed to be [keV]

keywords
        NH      [INPUT; default=1] atomic H I column density [cm^-2]
                * when keyword VION is set, IONABS() is called to deal
                  with partially ionized media.  In this case, NH is
                  assumed to be total H column, i.e., H I + protons
                * note that when NH=1, the function essentially returns
                  the absorption cross-section in units of [cm^2]
                * to set the "column" for UV/optical/IR, use the keyword
                  EBV, which defines the E(B-V) and gets passed down to
                  the routine FM_UNRED
                  - this means that the "feature" of getting the cross-section
                    out for NH=1 that is available for high-energy photons is
                    not available at wavelengths > 911.75 Ang
        fH2     [INPUT; default=0.] N(H2)/N(HI)
                * originally was set to a default of 0.26, the Galactic
                  average -- comes from the asymptotic value of
                  N(H2)/N(HI) computed with the 3 component model (cold
                  HI, warm HI, H2) of Bloemen, J.B.G.M. (1987, ApJ 322, 694)
                  -- but has now been changed to 0. explicitly set it to a
                  float number to include H2 cross-sections
                * if NoHeH is set to exclude H, fH2 is assumed to be 0
        He1     [INPUT; default=0.1*NH] neutral He column density [cm^-2]
        HeII    [INPUT; default=0.1*He1] ionized He column density [cm^-2]
                * if NoHeH is set to exclude He, both HeI and HeII cross-sections
                  will be excluded regardless of what the keywords He1 and
                  HeII are set to
        Fano    [INPUT] if set, the 4 strongest auto-ionizing resonances
                of He I are included; the shape of these resonances are
                given by a Fano profile (cf. Rumph, Bowyer, & Vennes 1994,
                AJ 107, 2108).
                * input wavelength grid between 190 A and 210 A should be
                  ~0.01 A for this to be useful
        ikev    [INPUT] if set, assumes that W are in units of keV
        vion    [INPUT] if set, uses Verner et al. (1996) cross-sections
                * calls IONABS()
        wam     [INPUT] if set, uses Wilms, Allen, & McCray cross-sections
                * NOT IMPLEMENTED YET -- if set, defaults to Balucinska-Church
                  & McCammon.
```

```
bam      [INPUT] if set, uses Balucinska-Church and McCammon updates
         to Morrison & McCammon cross-sections in 30 eV - 10 keV range
         * calls BAMABS()
mam      [INPUT] if set, uses Morrison and McCammon photoionization
         cross-sections in the 30 eV - 10 keV range
         * This is the default
         * VION takes precedence over
           WAM takes precedence over
           BAM takes precedence over MAM.
         * if MAM is explicitly set to 0 and none of VION, BAM, or WAM are
           set, then the original ISMEUV code is used from 43 AA longwards
noHeH    [INPUT] passed on unchanged to BAMABS() and IONABS()
         if set to any number other than 1 or 2, excludes H and He
         cross-sections in BAMABS (and ensures that even if fH2 and
         He1 are not zero, they are properly excluded within ISMTAU)
         * if set to 1, excludes only H
         * if set to 2, excludes only He
         * NOTE: if set, will supersede fH2, He1 and HeII keyword
           values in that H, H2, He1, He2 are all appropriately
           excluded
tauH1    [OUTPUT] optical depth due to H
tauH2    [OUTPUT] optical depth due to molecular H2
tauHe2   [OUTPUT] optical depth due to HeII
tauHeII  [OUTPUT] same as TAUHE2
         * both present here for backwards compatibility
tauHe1   [OUTPUT] optical depth due to HeI
         * NOTE: if VION is set, it computes H, HeI, and HeII
           cross-sections, so the parts of tauH1, tauHe1, tauHe2
           that overlap the relevant range get set to zero
tauX     [OUTPUT] optical depth in EUV and X-ray regime (0.03-10 keV)
tauTB    [OUTPUT] optical depth in hard X-ray regime (>10 keV)
icrstr   [I/O] passed without comment to IONABS()
EBV      [INPUT] color excess, E(B-V), passed on to FM_UNRED
         * default is 0
         * note that -ve values are not forbidden in FM_UNRED,
           but are ignored here because we don't want any
           confusion with the other optical depths
R_V      [INPUT] passed without comment to FM_UNRED
LMC2     [INPUT] passed without comment to FM_UNRED
AVGLMC   [INPUT] passed without comment to FM_UNRED
ExtCurve[INPUT] passed without comment to FM_UNRED
fmgamma  [INPUT] passed without comment to FM_UNRED
fmx0     [INPUT] passed without comment to FM_UNRED
fmc1     [INPUT] passed without comment to FM_UNRED
fmc2     [INPUT] passed without comment to FM_UNRED
```

```
        fmc3    [INPUT] passed without comment to FM_UNRED
        fmc4    [INPUT] passed without comment to FM_UNRED
                * note: all of these keywords are explicitly included
                  because FM_UNRED doesn't have an _EXTRA keyword
        _extra  [INPUT ONLY] pass defined variables to subroutines
                BAMABS: ABUND,VERBOSE
                IONABS: ABUND,IONFRAC,VFKYDIR,NCONSEC,DEM,LOGT,CHIDIR,EQFILE

restrictions
        requires BAMABS and IONABS (part of PINTofALE) and
        FM_UNRED and associated subroutines (from IDLastro)

history
        written by Vinay Kashyap (Feb97), based on
          ISMEUV.PRO (W.Landsman [Oct94], via ISM.C@cea-ftp.cea.berkeley.edu
          (Pat Jelinsky, Todd Rumph, et al.)) and ISMABS.PRO (VK; Dec91)
        changed default on HeII from 0 to 0.1*He1 (VK; Jan MM)
        changed default on NH from 1e20 to 1.0 (VK; MarMM)
        changed keyword KEV to IKEV; also for some reason H and He
          photoionization from ISMEUV had been cut off at the C edge,
          now corrected; added keywords WAM, BAM, and MAM, and call to BAMABS
          (VK; OctMM)
        bug correction: BAM was not filtering on energy; allowed setting
          MAM=0 to recover original ISMEUV calc (VK; Jun03)
        follow-up bug correction (VK; Jul03)
        added keyword NOHEH to streamline behavior of handling H and He
          cross-sections with BAMABS (thanks Christian Schneider; VK Feb07)
        added keywords TAUH1,TAUH2,TAUHE2,TAUHE1,TAUX,TAUTB, VION,ICRSTR,;
          changed all array index notation from old "()" to new "[]";
          **WARNING** changed behavior of FH2 -- new default is 0, not 0.26;
          added call to IONABS();
          (VK; May09)
        added call to FM_UNRED for wvl > 911.75 Ang (VK; Aug11)
        bug fixes (Jon Slavin; Sep13):
          /vion was double counting tauH,tauHe1,tauHe2
          (WARNING: changes behavior of tauH, tauHe1,tauHe2 on output)
        added keyword TAUHE2 (VK; Sep13)
```

B.97 ITOH_CH [/external]

PROJECT: CHIANTI

CHIANTI is an Atomic Database Package for Spectroscopic Diagnostics of Astrophysical Plasmas. It is a collaborative project involving the Naval Research Laboratory (USA), the University of Florence (Italy), the University of Cambridge and the Rutherford Appleton Laboratory (UK).

NAME

ITOH_CH

EXPLANATION

Calculates the relativistic free-free continuum using the fitting formula of Itoh et al. (ApJS 128, 125, 2000).

To calculate the continuum spectrum the user should specify whether the plasma structure is specified through a differential emission measure (DEM) curve or through emission measure (EM) values. For the former, the DEM values are specified through the DEM_INT keyword and for the latter the EM values are specified through the EM_INT keyword. If neither are specified EM_INT values of unity are assumed.

If DEM_INT values are specifed they must be specified on a temperature scale that has constant intervals in log10(T). EM_INT values do not have this restriction.

Note that EM_INT must be used for isothermal plasmas.

INPUTS

TEMP	Temperature (in K). If the keyword DEM_INT= is set, then the temperatures must be specified with a fixed spacing in logT. E.g., TEMP=10.^[4.0,4.1,4.2].
WVL	Wavelengths in angstroms. Can be a scalar or vector. If /keV is set, then WVL is assumed to contain energies in keV units.

OUTPUTS

INT	Free-free continuum emissivity in units of 10^-40 erg cm^3 / s / sr / Angstrom per unit emission

```
           measure [ integral(N_e N_H dh) in cm^-5 ]. If T is given as
           a 1-D array, then RAD will be output as a 2-D array,
           with one element for each temperature and wavelength
           (but also see SUMT).

           If the keyword /keV is set, then the units of INT will be
           10^-40 erg cm^3/s/sr/keV

OPTIONAL INPUTS

   DEM_INT   This should be the same size as TEMP and contain
             differential emission measure values. Specifying
             DEM_INT places a restriction on TEMP (see above). Note
             that DEM_INT values are multiplied by the factor dT
             within the routine, whereas EM_INT values are not.

   EM_INT    This should be the same size as TEMP and contain
             emission measure values. Note
             that DEM_INT values are multiplied by the factor dT
             within the routine, whereas EM_INT values are not. If
             neither EM_INT or DEM_INT are specified, then a EM_INT
             vector of same size as TEMP with values of unity is
             assumed.

   MIN_ABUND This keyword allows the specification of a minimum abundance,
             such that any elements with an abundance (relative to
             hydrogen) less than MIN_ABUND will not be included in the
             calculation. E.g., MIN_ABUND=1e-5.

KEYWORDS

   NO_SETUP By default the routine asks the user which ion balance
            and abundance files to use via pop-up widgets. If
            /no_setup is used then this data is taken from the common
            block.

   SUMT     The default is to output the intensity array as an array
            of size (nwvl x nT). Setting this keyword performs a sum
            over the temperatures to yield a vector of same size as
            the input wavelengths, thus producing the complete
            free-free spectrum.

   PHOTONS  Gives output emissivity in photon units rather than ergs.

   KEV      If set, then WVL is assumed to contain energies in keV units
```

```
            rather than wavelengths in Angstroms.

CALLS

   SETUP_ELEMENTS

PROGRAMMING NOTES

   In the Itoh et al. paper they state that their fitting formula is
   valid for ion charges (Z_j) of 1-28. The data-file they provide
   actually goes up to Z_j=30, and so you will see that the loop over
   z (see below) goes up to 30.

   There is no restriction on the elements for which the fitting
   formula is valid, and so all elements are allowed to be included
   (subject to their abundances being non-zero).

HISTORY

   Ver.1, 3-Dec-2001, Peter Young

   Ver.2, 22-May-2002, Peter Young
           Added MIN_ABUND optional input.

   Ver.3, 28-May-2002, Peter Young
           Corrected way in which DEM is handled.

   Ver.4, 9-Mar-2006, Peter Young
           Added /keV keyword.

   Ver.5, 10-Sep-2009, Peter Young
           Routine now checks that temperatures are evenly spaced
           when DEM_INT is specified. Also 0.1 dex spacing is no
           longer assumed.

   Ver.6, 6-Oct-2009, Peter Young
           Introduced EM_INT keyword for specifying emission measure
           values for isothermal plasmas.

   ver.PoA 10-Aug-2010, Vinay Kashyap
            a la 18-Feb-2005, LiWei Lin
           Commented out common block
           Added abund, ioneq, ioneq_logt keywords
           Replaced variable ioneq_t with ieq_logt, _extra
           Renamed routine itoh_ch
```

B.98 KABOOM [/util]

```
procedure       kaboom
        produces BOOMs and BANGs
        (to be used only to indicate catastrophic failure, OK?  don't overdo it)

syntax
        kaboom,boom,bang=bang,/beep,/flash,/help

parameters
        boom    [INPUT; default='KABOOM!!'] could be 'BIFF'.  or 'POW'.  could
                even be an array

keywords
        bang    [INPUT; default=42] number of "KABOOM"s to produce
        beep    [INPUT] if set, also produces beeps
        flash   [INPUT] if set, flashes the window
        help    [INPUT] prints usage and quits
        _extra  [JUNK] here only to prevent crashing the program

usage examples
        kaboom,boom,bang=bang,/beep,/flash,/help
        kaboom,'?',bang=100,/beep,/flash

side-effects
        makes an almighty mess of the plot window; may also make noises

history
        vinay kashyap (rewritten from memory of 1993 program)
        added keywords FLASH, HELP, and _EXTRA (VK; Nov98)
        improved color-scale setting for 24-bit consoles (VK; FebMMI)
        added a brief pause (VK; Apr04)
```

B.99 KARZAS_XS [/external]

```
PROJECT:  CHIANTI

     CHIANTI is an Atomic Database Package for Spectroscopic Diagnostics of
     Astrophysical Plasmas. It is a collaborative project involving the Naval
     Research Laboratory (USA), the University of Florence (Italy), the
     University of Cambridge and the Rutherford Appleton Laboratory (UK).

NAME

   KARZAS_XS()

EXPLANATION

   Outputs the photoionization cross-section at the wavelengths WVL for
   ionization of an N,L electron with ionization energy IONIZ_EN,
   calculated using the Karzas & Latter (ApJS 6, 167, 1961) formulation.
   The bound-free gaunt factor is derived from the tables of Karzas &
   Latter.

INPUTS

   WVL    Wavelengths (in angstroms) for which cross-sections are
          required (1-D array).

   N      Principal quantum number of the electron being removed in the
          ionization.

   L      The orbital angular momentum of the electron being removed in the
          ionization.

   IONIZ_EN  The ionization energy (in cm^-1) of the electron being
             removed.

OPTIONAL INPUTS

   PE      Allows the PE array from READ_KLGFB to be directly input to
           KARZAS_XS thus avoiding the need to read the K&L data
           repeatedly for many ions. Requires KLGFB to also be input.

   KLGFB   Allows the KLGFB array from READ_KLGFB to be directly input to
           KARZAS_XS thus avoiding the need to read the K&L data
           repeatedly for many ions. Requires PE to also be input.
```

OUTPUT

The photoionization cross-section for the ionization of the outer electron in units of mega-barns (Mb = 10^-18 cm^2) at the input wavelengths. E.g., for Fe XIII (ground configuration 1s2.2s2.2p6.3s2.3p2) it is the cross-section for the removal of the 3p electron.

CALLS

READ_KLGFB

HISTORY

Ver.1, 24-Jul-2002, Peter Young

B.100 KILROY [/util]

```
procedure       kilroy
        "kilroy was here"
        writes a little something to the screen every time it is called

syntax
        kilroy,here,ndot=ndot,dot=dot,say=say

parameters
        here    [I/O; optional] if given, is included in the output and
                is increased by 1 upon output.
                * cannot be a string

keywords
        ndot    [INPUT] number of dots to write out (default is 1)
        dot     [INPUT] symbol to use (default is ".")
        say     [INPUT] an extra word or whatever to print out
        _extra  [JUNK] here only to avoid crashing the program

history
        vinay kashyap (Aug 1996)
```

B.101 LAT2ARAB [/util]

```
function        lat2arab
                return arabic (decimal) equivalent of a latin numeral

syntax
        arabic=lat2arab(latin)

parameters
        latin   [INPUT; required] the latin numeral

keywords        NONE

restrictions
        * spaces, "."s, and anything that is not part of the
          roman numeral system (I V X L C D M) is ignored.
        * there is no reverse function, to go from arabic
          to latin, because that is not a unique transformation.
          e.g., 1995 could be either MDCCCCLXXXXV (the BBC way),
          or MVM, or all steps in between.
        * guaranteed to work only in simple cases.
          DON'T push the envelope!
        * CAVEATVSVATOR

history
        vinay kashyap (Nov95)
```

B.102 LEGALVAR [/util]

```
function        legalvar
        returns 1 or 0 depending on whether VAR is a legal IDL variable
        or not.  simply calls EXECUTE.  need to put this inside a subroutine
        to protect possible existing variables of same name in main routine.

syntax
        l=legalvar(vars,verbose=verbose)

parameters
        var     [INPUT; required] string of candidate variable names
                * if array, output will also be an array of 1s and 0s

keywords
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (MM.I)
        bug fix if input is scalar (VK; MM.X)
        added keyword verbose (VK; IMVIM.VIII)
```

B.103 LEVMARQ [/fitting]

procedure levmarq

Levenberg-Marquardt method, attempting to reduce the value chi^2 of a fit between a set of data points Y(X) with individual std.dev.s SIG, and a nonlinear function dependent of A parameters. The input array IFIT indicates by non-zero entries those components of A that should be fitted for, and by 0 entries those components that should be held fixed at their input values. The program returns current best-fit values for the parameters A and the value of CHISQ. The arrays COVAR and ALPHA are used as working space during most iterations. Supply a routine FUNCS(x,a,yfit,dYdA) that evaluates the fitting function YFIT, and its derivatives dY/dA wrt fitting parameters A@X. On the first call, provide an initial guess for parameters A, and set ALAMDA < 0 for initialization. If a step succeeds, CHISQ becomes smaller and ALAMDA decreases by a factor 10. If a step fails, ALAMDA grows by a factor 3. Call this routine repeatedly until convergence is achieved. Then, make one final call with ALAMDA=0, so that COVAR returns the covariance matrix, and ALPHA the curvature matrix. Those parameters held fixed will return zero covariances.

syntax

```
levmarq,x,y,a,chisq,funcs=funcs,sig=sig,ifit=ifit,alamda=alamda,$
jumpup=jumpup,jumpdn=jumpdn,ochisq=ochisq,alpha=alpha,covar=covar,$
bvec=bvec,svdthr=svdthr,trial=trial,yfunc=yfunc,/tiptoe,$
funcs=funcs,sig=sig,ties=ties, FUNCS_KEYWORDS
```

parameters

x	[INPUT; required]
y	[INPUT; required] size must match that of X
a	[INPUT; required]
chisq	[OUTPUT; required] the chi-sq statistic denoting degree of agreement between model and data

keywords

ifit	[INPUT] integer array of same size as A, with 0's indicating frozen parameters and 1's indicating thawed parameters * if IFIT does not match A for any reason, all parameters are assumed to be thawed
jumpup	[INPUT; default=3] factor by which to increase ALAMDA if current trial fails
jumpdn	[INPUT; default=0.1] factor by which to decrease ALAMDA if current trial succeeds
alamda	[I/O] if not defined on input or is -ve, initializes the procedure.

```
        ochisq  [OUTPUT] the old value of the chi^2
        alpha   [OUTPUT] if ALAMDA=0, the curvature matrix
        covar   [OUTPUT] if ALAMDA=0, the covariance matrix
        bvec    [OUTPUT] the BETA fitting-vector
        svdthr  [INPUT; default=1e-6] threshold for singular values of
                diagonal SVD matrix
        trial   [OUTPUT] current trial values of the parameters
        yfunc   [OUTPUT] best-fit Y(X;A)
        tiptoe  [INPUT] if set, forces small steps on A
        verbose [INPUT] controls chatter
        _extra  [INPUT] pass defined keywords to subroutines
                LMCOEFF: FUNCS, SIG
                ADJUSTIE: TIES
                FUNCS: whatever is needed

subroutines
        LMCOEFF
        SVDC
        SVSOL
        ADJUSTIE

history
        (C) Copr. 1986-92 Numerical Recipes Software =$j!]Y'1,).
        translated from MRQMIN.F to IDL by Vinay Kashyap (Oct98)
        changes: (i) matrix solution method from Gauss-Jordan elimination
          to Singular-Value Decomposition; (ii) if trial fails, ALAMDA
          increased by x{JUMPUP=3} instead of x10 and if trial succeeds,
          decreased by x{JUMPDN=0.1} to avoid obvious oscillations;
          (iii) call to COVSRT avoided; (iv) call to ADJUSTIE included;
          (v) no resetting chisq within procedure (VK; Oct98)
        added keyword YFUNC (VK; Dec98)
        allowed "inversion" of 1-element "matrices" bypassing
          SVDC and SVSOL (VK; Aug99)
        now returns correct chisq if all params are frozen (VK; JanMM)
        added keyword VERBOSE (VK; Nov04)
```

B.104 LIBMODEL [/fitting]

```
procedure       libmodel
        computes F(X;A) and dF/dA for a library of model functions.
        procedure written to be compatible with FIT_LEVMAR, and also
        with IDL's CURVEFIT, GHRS-IDL's WFIT, etc.

syntax
        libmodel,x,a,f,pder,anew,type=type,verbose=verbose,$
        missing=missing,/norm,/fwhm,betap=betap,angle=angle,vrot=vrot,$
        Xo=Xo,pbreak=pbreak,NH=NH,wvlar=wvlar,effar=effar,exptime=exptime,$
        dellam=dellam,X0=X0,angrise=angrise,angfall=angfall,ybase=ybase,$
        yoff=yoff

parameters
        x       [INPUT; required] points at which to generate models
        a       [INPUT; required] array of parameter values
                * should match the number expected from TYPE.
                * no checks are made to verify that the user is not shooting
                  hirself in the foot.
        f       [OUTPUT; required] output f=f(x;a)
        pder    [OUTPUT; optional] partial derivatives for each parameter
        anew    [OUTPUT; optional] return a new set of A in which the
                normalization parameters have all been adjusted to account
                for the supplied RENORM

keywords
        type    [INPUT; required] string array denoting the model functions
                to be called.
                * this should rightly be a parameter, because it is
                  a very necessary part of the program.  however, most
                  fitting functions do not allow that degree of flexibility
                  and so we are forced to use this hack.
                * some models may be multiplicative (e.g., 'absorb'), and
                  their location in the TYPE array matters.  a multiplicative
                  model multiplies the model that is defined after it.  so
                  to apply the same absorption model to a powerlaw and a
                  gaussian, set
                    TYPE=['absorb','power','absorb','gauss']
                  and then use TIES=['a6=a1'],FREEZE=[6] to make sure the same
                  absorption is applied to both components
        betap   [I/O] keyword passed w/o checking to MK_LORENTZ and MK_SLANT
        angle   [I/O] keyword passed w/o checking to MK_SLANT
        vrot    [I/O] keyword passed w/o checking to MK_ROGAUSS
        x0      [I/O] keyword passed w/o checking to MK_POWLAM,MK_BKNPOWER
```

```
        pbreak  [I/O] keyword passed w/o checking to MK_POWLAM,MK_BKNPOWER
        phase   [I/O] keyword passed w/o checking to MK_SINUSOID
        angrise [I/O] keyword passed w/o checking to MK_LCECLIPSE
        angfall [I/O] keyword passed w/o checking to MK_LCECLIPSE
        ybase   [I/O] keyword passed w/o checking to MK_LCECLIPSE
        yoff    [I/O] keyword passed w/o checking to MK_LCECLIPSE
        verbose [INPUT; default=0] controls chatter
        renorm  [INPUT; default=1] renormalize the output by multiplying
                by this factor
                * the adjusted normalizations for each component that
                  correspond to this renormalization are returned in ANEW

        _extra  [INPUT] pass defined keywords to subroutines
                MK_GAUSS: MISSING, FWHM, NORM
                MK_LORENTZ: BETAP, MISSING, NORM
                MK_SLANT: ANGLE, BETAP, MISSING, NORM
                MK_ROGAUSS: VROT, FWHM, NORM, MISSING
                MK_POWLAM: X0,PBREAK,NH,WVLAR,EFFAR,EXPTIME,DELLAM
                MK_BKNPOWER: X0,NOBREAK,VERBOSE
                MK_POLY1D: X0
                MK_SINUSOID: PHASE, MISSING
                MK_SINE: MISSING
                MK_LCECLIPSE: ANGRISE, ANGFALL, YBASE, YOFF
                MK_ABSORB: FH2,HE1,HEII,FANO,IKEV,WAM,BAM,MAM,NOHEH,ABUND
                MK_EFOLD: VERBOSE
                MK_BBANG: VERBOSE
                MK_BBKEV: VERBOSE
                MK_RECIPROCALPOW: VERBOSE
                MK_SPLINE1D: XLOC,USEINIT,TENSION,VERBOSe

subroutines
        MK_GAUSS
        MK_LORENTZ
        MK_SLANT
        MK_ROGAUSS
        MK_POWLAM
        MK_BKNPOWER
        MK_POLY1D
        MK_LCECLIPSE
        MK_ABSORB
        MK_EFOLD
        MK_BBANG
        MK_BBKEV
        MK_RECIPROCALPOW
        MK_SINUSOID
```

```
        MK_SINE

history
        vinay kashyap (Aug01; based on X3MODEL and MK_3MODEL)
        added call to MK_SINUSOID (VK; Sep04)
        added call to MK_LCECLIPSE, allowed return of extra parameters
          via keywords (VK; May07)
        added call to MK_BKNPOWER via "pow" and "pe"; added MK_ABSORB
          via "absorb"; allowed for multiplicative as well as additive
          models (VK; Jul08)
        added call to MK_EFOLD via "ef", "efold", and "exp" and to
          MK_BBANG and MK_BBKEV via "bbang" and "bbkev" (VK; Aug08)
        added call to MK_RECIPROCALPOW via "reci/pow" and MK_SPLINE1D
          via "spline=KNOTS"; added parameter ANEW and keyword RENORM
          (VK; Sep08)
        added call to MK_SINE (VK; Jan13)
```

B.105 LICOSPEC [/]

```
procedure       licospec
        compute the contributions of line and continuum
        emissivities over a specified wavelength grid.

syntax
        licospec,wgrid,lspec,cspec,at_logT,lrf=lrf,lstr=lstr,cstr=cstr,$
        lemis=lemis,cemis=cemis,abund=abund,lcthr=lcthr,constr=constr,$
        ldbdir=ldbdir,cdbdir=cdbdir,ceroot=ceroot,verbose=verbose,$
        pres=pres,logP=logP,n_e=n_e,chifil=chifil,chidir=chidir,$
        eqfile=eqfile,logT=logT,DEM=DEM,effar=effar,wvlar=wvlar,$
        type=type,/norm,/fwhm,betap=betap,angle=angle,vrot=vrot

parameters
        wgrid   [INPUT; required] wavelength bin boundaries over
                which to compute line and continuum contributions
                * must be in same units as the wavelengths stored
                  in LDBDIR, i.e., Angstroms
        lspec   [OUTPUT; required] line emission at each wavelength bin
        cspec   [OUTPUT; required] continuum emission at each wavelength bin
                * note that these are essentially spectra, and include
                  the effects of specified abundance, DEM, and any input
                  RMFs, LRFs, or effective areas.
        at_logT [INPUT] if set to a scalar, assumes a delta-function
                emission measure to compute the appropriate fluxes;
                the default is to assume a DEM that covers the range
                4..8 in logT (but note that any arbitrary DEM can be
                defined using keywords LOGT and DEM).
                * NOT TESTED

keywords
        lrf     [INPUT] if scalar, the width of the line response function
                NOTE: the function used to define the LRF can be set
                using the keyword TYPE (e.g., TYPE='gauss', TYPE='beta=2.5')
                * if float, the width in the same units as WGRID
                * if integer, the width in number of bins
                * if RMF structure (e.g., output of RD_OGIP_RMF()), use
                  it to convolve the output and place in channels defined
                  by WGRID indices (i.e., if LRF is defined as an RMF,
                  it _must_ match the wavelength grid)
        lstr    [I/O] line emissivity structure out of RD_LINE()*FOLD_IONEQ()
                * if defined on input, RD_LINE() and FOLD_IONEQ() will not
                  be called unless bounds of WGRID spill out, in which case
                  the user will be asked what to do
```

```
                * unlike LEMIS below, abundances are _not_ included
                  (unless in case of APED)
        cstr    [I/O] continuum emissivity structure out of RD_CONT() --
                * if defined on input, RD_CONT() will not be called unless
                  bounds of WGRID spill out, in which case the user will be
                  asked what to do
        ldbdir  [INPUT; '$CHIANTI'] line emissivity database directory
        cdbdir  [INPUT; '$CONT'] continuum emissivity database directory
        ceroot  [INPUT; 'cie'] prefix of files containing the continuum
                emissivities
        abund   [INPUT] element abundances to use while computing the
                continuum emissivities, the line and continuum fluxes,
                combining line emissivities into wavelenth bins, etc.
                * if not defined, calls GETABUND(!ABREF)
                * if !ABREF is not defined, adopts Grevesse et al.
        lemis   [OUTPUT] line emissivities recast to be on the input
                wavelength grid, i.e., an array of size N(LOGT)xN(WGRID)
                and with units [1e-23 ergs cm^3/s]
                * the output line emissivities _include_ the specified
                  ion balances and abundances
        cemis   [OUTPUT] continuum emissivities recast to be on the input
                wavelength grid, i.e., an array of size N(LOGT)xN(WGRID)
                and with units [1e-23 ergs cm^3/s]
                * NOTE: the /Ang usually present in continuum emissivities
                  is missing here
        lcthr   [INPUT] if set, combines the continuum emissivity from all
                the bins with LSPEC/CSPEC less than LCTHR and returns it
                in CONSTR
                * if negative, then combines all bins with CSPEC < LCTHR
        constr  [OUTPUT] continuum emissivity structure with the emissivities
                marginalized over wavelength
       allah   [INPUT] if set, no bounds of WGRID are not checked against
               LSTR and the structure is used as is
        verbose [INPUT] controls chatter
        _extra  [INPUT ONLY] pass defined keywords to subroutines
                RD_LINE: PRES, LOGP, N_E
                FOLD_IONEQ: CHIFIL, CHIDIR, EQFILE
                RD_CONT: PRES, LOGP, N_E, ABUND
                LINEFLX: LOGT, DEM, ABUND, EFFAR, WVLAR
                LIBMODEL: TYPE, NORM, FWHM, BETAP, ANGLE, VROT

example
        ;       make a wavelength grid
        wgrid=findgen(2001)*0.001+18. & lrf=0.02 & type='beta=2.5'
        ;       call LICOSPEC
```

```
        licospec,wgrid,lspec,cspec,verbose=10,lstr=lstr,cstr=cstr,$
        lcthr=0.1,constr=constr,lemis=lemis,cemis=cemis,n_e=1e10,$
        lrf=lrf,type=type
        ;       plot line to continuum flux ratio
        plot,wgrid,lspec/cspec,/ylog,$
        xtitle='wavelength [Ang]',ytitle='line-to-continuum'
        ;       plot line and continuum integrated emissivities
        tmpl=dblarr(81) & for i=0,80 do tmpl[i]=total(lemis[i,*])
        tmpc=dblarr(81) & for i=0,80 do tmpc[i]=total(cemis[i,*])
        plot,lstr.LOGT,tmpl,/yl & oplot,cstr.LOGT,tmpc,thick=3,line=2
        ;       integrated emissivities over a sub range
        oo=where(lspec/cspec lt 0.1)
        tmpl=dblarr(81) & for i=0,80 do tmpl[i]=total(lemis[i,oo])
        tmpc=dblarr(81) & for i=0,80 do tmpc[i]=total(cemis[i,oo])
        plot,lstr.LOGT,tmpl,/yl & oplot,cstr.LOGT,tmpc,thick=3,line=2

subroutines
        CONV_RMF [RD_OGIP_RMF]
        FOLD_IONEQ [RD_IONEQ [READ_IONEQ], WHEE]
        GETABUND
        HASTOGRAM [KILROY]
        LIBMODEL [MK_GAUSS, MK_LORENTZ, MK_ROGAUSS [MK_GAUSS], MK_SLANT]
        LINEFLX [GETABUND, WHEE]
        MID2BOUND
        RD_CONT [SETSYSVAL [LEGALVAR], SYMB2ZION [LAT2ARAB]]
        RD_LINE [INICON, SETSYSVAL [LEGALVAR], SYMB2ZION [LAT2ARAB]]
        REBINW [FINDEX]
        SETSYSVAL [LEGALVAR]
        IS_KEYWORD_SET

history
        vinay kashyap (Dec2003)
        added keywords LEMIS,CEMIS,LCTHR,CONSTR,ABUND,LDBDIR,CDBDIR,CEROOT;
          corrected bug with CSTR.CONT_INT being degraded in repeated calls;
          now -asks- if input wavelength range has changed and LSTR/CSTR do
          not span this range (VK; Jan2004)
        cleaned up behavior of CSTR.CONT_INT for compatibility with RD_CONT();
          now output LEMIS and CEMIS add up in photon space (VK; Apr2004)
       added ALLAH keyword to facilitate use of ZRESP keyword in
        SOLAR_TRESP() (LL; Sep2005)
        updated for IDL5.6 keyword_set([0]) behavior change for vectors
          (VK; 20Mar2006)
```

B.106 LIKELI [/stat]

```
function        likeli
        returns p(D|M), the likelihood of observing the data for the
        given model.

        the default action is to return exp(-chi^2/2)

syntax
        prob=likeli(data,model,derror,dsigma=dsigma,ulim=ulim,$
        /binom,/chi2,/rcfhi,/cash,/castor,softlim=softlim)

parameters
        data    [INPUT; required] observed data
        model   [INPUT; required] predicted model or model parameters
        derror  [INPUT] errors on DATA
                * if specified, overrides keyword DSIGMA

keywords
        dsigma  [INPUT] errors on DATA
                * if not specified, taken to be 1+sqrt(abs(DATA)+0.75)
                * used only if DERROR is not present
        ulim    [INPUT] long-integer array specifying which elements of
                DATA are upper limits (1: UL, 0: not)
                * applies only to CHI2 and RCHI
        cash    [INPUT] if set, returns the Cash (19??, ApJ 228, 939) statistic
        castor  [INPUT] if set, returns the Castor correction to Cash
        binom   [INPUT] if set, likelihood is computed as
                    MODEL(0)^(d1)*(1.-MODEL(0))^(d0)
                where
                    d1=total(DATA(where(DATA ge BINOM))) and
                    d0=total(DATA(where(DATA lt BINOM)))
        chi2    [INPUT; default] if set, returns -alog(p(D|M))
        rchi    [INPUT] if set, returns alog(p(D|M))/n_elements(D)
                * priority: CASH supercedes CASTOR supercedes
                  BINOM supercedes CHI2 supercedes RCHI
        softlim [INPUT] if set, then likelihood is unaffected when
                model values are below data values, and drops as a
                Gaussian centered on SOFTLIM*DSIGMA with error DSIGMA
                otherwise
                * if not set, upper limits are taken to be hard
                  limits, i.e., likelihood drops to 0 if model
                  values exceed data values
        _extra  [INPUT] junk -- here only to avoid crashing the program
```

```
history
        vinay kashyap (Mar 97)
        added keywords CASH and CASTOR (VK; Jul98)
        changed keyword name SIGMA to DSIGMA (VK; MMaug)
        had forgotten to actually include CASH and CASTOR in call (VK; May02)
        added parameter DERROR as a dupe for DSIGMA (VK; Jun05)
        BUGFIX to not crash if all DATA are upper limits (VK; Sep05)
        added keyword SOFTLIM (VK; May06)
        bug correction re SOFTLIM model estimate (VK; Jul07)
        corrected output to be same as all the others when /BINOM,/CHI2
          is set or when /RCHI is set but not /CHI2 (VK; Apr08)
        fixed a lower bound of 1e-30 for model intensities for CASH per
          discussion with Jan-Uwe (VK; Aug08)
```

B.107 LINE_CHIANTI [/mkemis]

```
FUNCTION        line_chianti
        returns a 2D array containing line cooling emissivities at a set of
        temperatures for all available spectral lines (in a CHIANTI style
        database [http://wwwsolar.nrl.navy.mil/chianti.html]) for the
        specified element at the specified pressure or density
        [1e-23 erg cm^3 s^-1] (NTEMP,NANG)
        NOTE: Ion Balance is NOT included!!!

SYNTAX
        fx=line_chianti(elem,pres,logT,ang,trans,kev=kev,ikey=ikey,jkey=jkey,$
        econf=econf,wmn=wmn,wmx=wmx,n_e=n_e,istate=istate,/tex,chidir=chidir)

PARAMETERS
        elem    [INPUT; required] name of element e.g., "He", "C", etc.
                * may specify ionic state (e.g., 'Fe XXII')
                * if an array, only the first element is looked at
        pres    [INPUT; default: 1e15 cm^-3 K] pressure at which to compute
                level populations
        temp    [INPUT/OUTPUT] log(Temperature[K]) at which to compute the
                line fluxes.  if not given, then TEMP=4.0+FINDGEN(81)*0.05
                * best results if evenly spaced in log(T)
        ang     [OUTPUT] all the wavelengths [Angstrom]
        trans   [OUTPUT] 2D array of transitions --
                trans[0,*] lower levels; trans[1,*] upper levels

KEYWORDS
        kev     [OUTPUT] all the line energies (same as ANG, but in [keV])
        ikey    [OUTPUT] ionic state of element producing the line at ANG
        jkey    [OUTPUT] ionic state that matters to ion balance
        econf   [OUTPUT] 2D string array of e-configurations --
                econf[0,*] lower levels; econf[1,*] upper levels
        wmn     [INPUT; default: 0 Angstrom] minimum value of wavelength
                to include in output
        wmx     [INPUT; default: 900 Angstrom] maximum value of wavelength
                to include in output
        n_e     [INPUT] electron density [/cm^3]
                * OVERRIDES values determined using PRES and TEMP
        istate  [INPUT; default: ALL; overridden by ELEM] limit calculations
                to specified ionic states (1=neutral, 2=singly ionized, etc.;
                can be an array)

        _extra  [INPUT ONLY] used to specify
                * /TEX -- TRANS in TeX format (1) or not (0)
```

```
                * CHIDIR -- path name to the CHIANTI installation

RESTRICTIONS
        * uses the CHIANTI database
        * requires subroutines:
          -- GET_IONLIST
          -- SYMB2ZION [LAT2ARAB]
          -- RD_CHIANTI [READ_WGFA2(%), READ_ELVLC(%), READ_SPLUPS(%), READ_IP(%)]
          -- POPSOL [DESCALE_UPS(%), NENH]
          -- TRANSLABEL
          -- KILROY
          -- IS_KEYWORD_SET
          (%): CHIANTI subroutine, used as is

HISTORY
        vinay kashyap (Nov96)
        removed ion balance, changed input keyword CONF to output ECONF,
          changed mult. factor from 1e30 to 1e23 (VK; Dec96)
        added level excitation check (VK; Jun97)
        corrected bug with TEMP specification; forced ELEM to be scalar
          (VK; Nov98)
        changed call from POPLEV to POPSOL (VK; SepMM)
        added keyword JKEY; corrected calls to modifications of GET_IONLIST,
          RD_CHIANTI, and POPSOL to account for dielectronic recombination
          lines; converted syntax to IDL5 (VK; OctMM)
        bug correction: CHIDIR was not being passed on; caught and
          filtered -ve population levels (VK; DecMM)
        bug correction: transition labels were being filtered on wavelength
          twice, once within TRANSLABEL and once here.  now forced the range
          in TRANSLABEL to be ignored (VK; Jun01)
        bug correction: was failing if ELEM contained ION info (VK; May02)
        modified call to POPSOL, per CHIANTI 4.2 changes (VK; Apr04)
        updated for IDL5.6 keyword_set([0]) behavior change for vectors
          (VK; 20Mar2006)
        made loops go over long integers (VK; 11Jun2010)
```

B.108 LINEFLX [/]

```
function        lineflx
        computes observeable counts [ct/s] or flux [ergs/s] from spectral
        line for specified DEM by simple trapezoidal integration over
        temperatures.
        if effective area is not specified, output will be [ph/s/cm^2]
        or [ergs/s/cm^2]

        input line emissivities are in [1e-23 ergs cm^3/s].  these include
        ionization balance, but not (necessarily) abundances.  the line
        emissivities are multiplied by the differential emission measure
        and element abundance to get fluxes, then divided by the line energy
        to get photon fluxes, and multiplied by the supplied effective areas
        to get count rates.  no effort is made to determine line profiles
        or to include spectral responses (beyond the effective areas).

syntax
        fx=lineflx(line,logT,wvl,Z,DEM=DEM,/temp,abund=abund,/noph,$
        nhne=nhne,effar=effar,wvlar=wvlar,/ikev,/noabund)

parameters
        line    [INPUT; required] array of line cooling emissivities
                in units of 1e-23 erg cm^3/s
                * if 2D array, LINE==LINE(logT,WVL)

                * WARNING: will return garbage if given 1D array LINE(WVL)
                  (or even 2D array LINE(1,WVL))
                  use a for-loop to handle such a case

                * WARNING: will get converted to 1-element vector if input
                  as scalar
        logT    [INPUT; required] array of log10(Temperature [K]) at
                which emissivities are given.
                * array size MUST match that of LINE
                * if not regularly gridded, remember to set REGRID!
        wvl     [INPUT; required] wavelength(s) [Angstrom] of lines at
                which LINE is given
                * array size MUST match LINE
        Z       [INPUT] atomic number of element that generates each WVL
                * if Z has less elements than WVL, Z is ignored
                * if Z is to be ignored, all lines are assumed to be from
                  same element, and abundance values are ignored

keywords
```

```
        DEM     [INPUT] Differential Emission Measure at each T [cm^-5/logK]
                * default is a constant=1e14!!
                * if array size does not match that of LOGT, then DEM is
                  assumed to cover the same range and is linear interpolated
                * if defined at only 1 temperature, assumed to be EM [cm^-5]
        temp    [INPUT] if set, assumes that logT is actually in T [K], and
                that DEM is given in units of [cm^-5/K]
                * ignored if logT is defined at only one bin
        abund   [INPUT] abundances relative to H (abund(0)=1)
                * abund(Z-1) contains the abundance for element Z
                * if array size is smaller than the largest present Z,
                  the last element is replicated to fill the gap
                * default: Anders & Grevesse (1989)
        noph    [INPUT] if set, does not do the conversion to [ph] from [ergs]
        nhne    [INPUT] the ratio of N(H)/n_e needed to complete the
                intensity calculation
                * if not set, assumes 0.83, which is the limiting value
                  for high-temperature plasma with approximately cosmic
                  abundances
                * if array and size does not match LOGT, then uses only
                  the first element
                * if Z is not defined, or is 1, then it is **IGNORED**
                * see discussion in the documentation of POPSOL()
        effar   [INPUT] effective area [cm^2]
                * if not set, assumed to be 1 cm^2
        wvlar   [INPUT] wavelengths at which effective area is defined.
                * if not set, assumed to be WVL
                * array size MUST match that of EFFAR
        ikev    [INPUT] if set, assumes that WVL and WVLAR are in [keV],
                NOT [Angstrom]
        noabund [INPUT] if set, ABUND values are ignored in the calculations,
                same as though they (or Z) had been set to 1
                * but NHNE gets applied
        regrid  [INPUT] if set, assumes that LOGT is irregularly sampled
                and regrids it (and LINE) to a regular grid
                * NOT IMPLEMENTED YET
        _extra  [INPUT] junk -- here only to prevent crashing the program

subroutines
        GETABUND
        WHEE

usage examples
        * f=lineflx(rd_line(logP=20,wvl=w,logT=T,/allah),T,w,dem=10.D^(5*t))
        * fl=rd_line(wr=16.776,wvl=w,logT=T,/allah) & t=10.^(t)
```

```
          f=lineflx(fl,T,w,dem=10.D^(5*alog10(t)),/temp)

history
        vinay kashyap (Nov96)
        added keywords EFFAR and WVLAR (VK; Jan96)
        added call to WHEE, removed call to KILROY,
          ensured DEM stretches to fit logT, added case of EM masquerading
          as DEM, removed the [/sr] from the output (VK; Feb97)
        changed integration style; restructured to ignore effective area
          if not supplied; added keyword REGRID (VK; Apr97)
        corrected ln v/s log10 bug (VK; Jun97)
        added keyword NOPH; bug correction -- dlogT should in in base 10
          (VK; Jan98)
        changed keyword KEV to IKEV (VK; DecMM)
        EFFAR and WVLAR ignored if set to 0 (VK; JanMMI)
        added keyword NHNE (VK; Jun02)
        added keyword NOABUND (VK; Apr03)
        changed bad input check from LINE(0)=-1 to total(LINE)=-N(LINE)
          (VK; Feb06)
        changed back integration style to simple from trapezoidal (VK; Mar06)
        changed check for whether WVL are given (VK; Jun07)
        bug fix: was including 0.83 factor even when continuua and broad-band
          emissivities were being used [thx P.Testa]; now anything with Z=1
          will exclude the NHNE correction -- for H lines, fold the NHNE factors
          into the emissivities manually (VK; Dec10)
```

B.109 LINEID [/]

```
function        lineid
        allows identification of spectral features and returns ID
        information in a structure (see structure variable IDHLP
        for complete description of fields)

syntax
        id=lineid(lamda,spec,elem,wrng=wrng,wshift=wshift,locmax=locmax,$
        best=best,topX=topX,batch=batch,stor=stor,oldid=oldid,omatch=omatch,$
        markw=markw,feature=feature,verbose=verbose,$
        n_e=n_e,logP=logP,pres=pres,dbdir=dbdir,/allah,chifil=chifil,$
        eqfile=eqfile,chidir=chidir,dem=dem,abund=abund,effar=effar,$
        wvlar=wvlar,sigmay=sigmay,xsize=xsize,ysize=ysize,wid=wid,$
        dynrng=dynrng,markp=markp,marks=marks,marko=marko,$
        /nuthin,/noday,/notime,/nouser,/nopack,stacol=stacol,stasiz=stasiz,$
        stathk=stathk)

parameters
        lamda   [INPUT; required] wavelength(s) at which spectrum is defined
                * interactive if LAMDA is a multi-element array, batch mode
                  otherwise (also see keyword BATCH)
        spec    [INPUT; optional] the spectrum
                * default is 10+random
                * if SPEC is a character string or array, taken to be ELEM
                * if LAMDA and SPEC do not match in size, taken to be atomic
                  numbers
        elem    [INPUT; optional] confine matches to this particular element
                * overrides anything inferred "via" SPEC

keywords
        wrng    [INPUT] interval in which to search for matches
                * default is WRNG=0.01
                * if scalar and between 0 and 1, look for matches in the range
                        [W-W*WRNG,W+W*WRNG]
                * if scalar and > 1, or scalar and < 0, abs(WRNG) is assumed
                  to be the actual step size and is used as is, i.e.,
                        W+abs(WRNG)*[-1,1]
                * if 1-element vector, then range is W+abs(WRNG(0))*[-1,1]
                * if multi-element vector, then [W-WRNG[0],W+WRNG[1]]
                * NOTE: in interactive mode the range may be altered, but
                  the >behavior< of the range is set by the initial values.
                * also note that WRNG is not changed on output.
        wshift  [INPUT] amount by which to shift the wavelength scale
                * WRNG will be reset to WRNG+WSHIFT
```

```
locmax  [INPUT] if set, finds matches at all local maxima
        * search window width (2*abs(LOCMAX)+1)>3
best    [INPUT] if set, returns "best" match -- the strongest
        possible nearest line
        * obviously, there is weighting involved
        * BEST.GE.0: pick largest of Flx[w]/(w-w0)^BEST
        * otherwise: return the nearest match(es)
topX    [I/O; default: 20] returns at most TOPX matches
        * MUST be scalar
        * in interactive mode, presents only TOPX for matching choice
        * if 0 < TOPX < 1, uses this as a >threshold<, selecting only
          those lines whose theoretical predicted fluxes are this
          fraction or higher than the maximum flux in the set.
        * if TOPX < 0, the integer value is used as the upper limit
          to number of matches, and the fractional value is used as
          the threshold.
        * may get altered within LINEID_MENU
        * examples:
          TOPX = 0 brings up 20 matches
          TOPX = 10 brings up 10 matches
          TOPX = 0.6 brings up all lines with fluxes > 0.6 of max
          TOPX = -40.01 brings up 40 lines with fluxes > 1% of max
batch   [INPUT] if set, operates in batch mode (i.e., no widget-based
        selections)
        * batch mode is also forced if:-
          o N_ELEMENTS(LAMDA)=1
          o LOCMAX is set
          o TOPX is set to 1
          o !D.NAME is not 'X'
stor    [I/O] if present in call, will store the output of RD_LINE and
        FOLD_IONEQ as a structure.  if a structure, then MUST be of
        the form {LINE_INT,LOGT,WVL,Z,ION,DESIG,CONFIG}, because
        then RD_LINE and FOLD_IONEQ will be bypassed!
        * use with caution!!
oldid   [INPUT] if set and is in the same format as the output of
        LINEID, starts off from this point.  i.e., things can be
        added or modified from here on.
omatch  [INPUT] if OLDID is given *and* OMATCH is set, forces each
        wavelength selection to match the nearest matched wavelength
        in OMATCH.
        * if OMATCH < 0, won't verify, but will force a match if
          new location is within abs(OMATCH), and will force a new
          match otherwise
        * if OMATCH > 0, will ask
markw   [INPUT] if set, places marks on the plot at these wavelengths
```

```
        feature [INPUT] labels for MARKW
                * passed with minimal modification to PICKRANGE
        verbose [INPUT] controls chatter
        wrange  [IGNORED] keyword for RD_LINE, will be trapped and ignored.
        temp    [IGNORED] keyword for LINEFLX, will be trapped and ignored.
        _extra  [INPUT ONLY] pass predefined input keywords to subroutines:
                pres    (RD_LINE) pressure [cm^-3 K]
                logP    (RD_LINE) log10(pressure [cm^-3 K])
                n_e     (RD_LINE) electron density [cm^-3]
                dbdir   (RD_LINE) line emissivity database
                allah   (RD_LINE) which elements to choose
                chifil  (FOLD_IONEQ) set to read CHIANTI ion balance files
                eqfile  (FOLD_IONEQ) name of ion balance file
                chidir  (RD_IONEQ) location of CHIANTI installation
                DEM     (LINEFLX) Differential Emission Measure distribution
                abund   (LINEFLX) abundances
                effar   (LINEFLX) effective area [cm^2]
                wvlar   (LINEFLX) wavelength grid for EFFAR
                sigmaY  (PICKRANGE) error-bars on SPEC
                xsize   (PICKRANGE) window size
                ysize   (PICKRANGE) window size
                wid     (PICKRANGE) window ID
                dynrng  (PICKRANGE) dynamic range in log scale
                markp   (PICKRANGE) PSYM for (FEATURE,MARKW)
                marks   (PICKRANGE) SYMSIZE for (FEATURE,MARKW)
                marko   (PICKRANGE) ORIENTATION for (FEATURE,MARKW)
                oxr     (PICKRANGE) final output XRANGE
                oyr     (PICKRANGE) final output YRANGE
                nuthin  (STAMPLE) don't stamp the plots
                noday   (STAMPLE) don't stamp the plots with day
                notime  (STAMPLE) don't stamp the plots with time
                nouser  (STAMPLE) don't stamp the plots with username
                nopack  (STAMPLE) don't stamp the plots with package name
                stacol  (STAMPLE) stamp color
                stasiz  (STAMPLE) stamp size
                stathk  (STAMPLE) stamp thickness

description
        read in database of line cooling intensities and for each observed
        wavelength which requires a match, find the set of matches.  if
        interactive, allow selection of multiple matches.  once done,
        reorganize into structure of
        {WVL: observed wavelength;
          {corresponding
            WVL: matched wavelengths,
```

```
            Z: atomic numbers,
            ION: ionic states,
            LABL: e-configurations or whatever is available,
            FLUX: theoretical fluxes for all new or modified matches
                (fluxes in OLDID are not touched unless ID is changed),
            FLUXERR: errors on FLUX (set to zero),
            LOGT: temperatures at which emissivities are defined
            EMIS: line emissivities for all IDs (ion-balance included)
            NOTES: scribbles about this ID, for human eyes only
          }
        }

restrictions
        requires IDL v5 or greater
        interactive selections require X-windows
        requires subroutines
            RD_LINE [KILROY, SYMB2ZION [LAT2ARAB]]
            FOLD_IONEQ [WHEE, GETLOCMAX, RD_IONEQ [READ_IONEQ]]
            LINEFLX [WHEE]
            GETLOCMAX
            PICKRANGE
            LINEID_MENU [CAT_ID]
            CREATE_STRUCT
            KABOOM
            INICON
            STR_2_ARR

usage examples
        * lid=lineid(12.3985/7.83,['Fe','Ni'],wr=0.1*[1,1],topX=50)
        * stor=1 & lid=lineid(x,y,wr=[1],/allah,logP=20,stor=stor)
          lid=lineid(x,y,wr=[1],/allah,logP=20,stor=stor)
        * lid=lineid(x,y,'Fe',logP=20,econf=0)

history
        vinay kashyap (Dec96)
        bug corrections: ion balance in STOR, format sizes in menu input;
          added ID history info to PICKRANGE and menu; added keyword OLDID
          (VK; Feb97)
        added keyword OMATCH; allowed expeditious return when no matches
          found (VK; Jun98)
        modified behavior of STOR to eliminate excess variables (VK; Oct98)
        added LOGT and EMIS (EMIS includes ion balance) to the output; bug
          correction re STOR not being specified; corrected bug with
          BEST actually returning the worst; added keywords BATCH, MARKW,
          FEATURE; added calls to KABOOM, INITSTUFF, STR2ARR; TOPX now acts
```

```
  as threshold too; added support for multiple grating orders
  (VK; Nov98)
changed behavior of WRANGE; added renormalization of fluxes; bug fix
  older matches not echoing in menu window; added field FLUXERR to
  ID structure (VK; Dec98)
added field NOTES to ID structure (VK; Mar99)
modified ion balance calcs (VK; 99May)
changed call to INITSTUFF to INICON (VK; 99May)
changed to full IDL5+ compatibility ("()"->"[]"); stor does not
  have to be defined on input, just be present in the call; added
  dummy keywords WRANGE and TEMP (VK; AugMM)
added keyword VERBOSE, and now doesn't default to random if NX eq NY+1
  (i.e., bin boundaries are given rather than mid-bin values: VK; Nov01)
changed call to STR_2_ARR from STR2ARR (VK; AprMMV)
```

B.110 LINEID_MENU [/]

```
function        lineid_menu
        function to select matches to wavelengths and return position
        indices of selected matches.

syntax
        oo=lineid_menu(menu,w0,w,f,wshift,topX,wrange,cord,code,scratch,$
        wvls=wvls,idwvls=idwvls,gwvl=gwvl,renorm=renorm)

parameters
        menu    [INPUT] list of items to choose from; string array
        w0      [INPUT] wavelength that needs match
        w       [INPUT] wvls of matched lines (must be of same size as MENU)
        f       [INPUT] fluxes of all matched lines (same size as MENU)
        wshift  [I/O] wavelength shift
        topX    [I/O] maximum number of matches
        wrange  [I/O] interval in which to search for matches
        cord    [I/O] comma-separated list of grating orders
        code    [OUTPUT] 0: NEXT WAVELENGTH
                         1: QUIT LINEID
                         2: REPEAT FOR NEW PARAMETERS
        scratch [OUTPUT] human readable notation

keywords
        wvls    [INPUT] wavelengths which are previously identified
        idwvls  [INPUT] structure describing IDs
        gwvl    [INPUT] grating-order corrected wavelengths
        renorm  [INPUT] renormalization factor

restrictions
        * widget based
        * dedicated subroutine to LINEID (written simply to avoid
          making LINEID too large to comprehend)
        * no error checking is done.  it is assumed that this function
          is called from LINEID, and naught else.
        * requires subroutines
          -- CAT_ID
          -- CREATE_STRUCT

warning
        * here be hippogrif
        * don't even <<think>> about understanding this program

history
```

```
author of this hack prefers anonymity, thank you very much (Dec96)
some cosmetic surgery, added button STATUS and call to CAT_ID (Feb97)
corrected bug with reordering (Jul98)
added CORD, GWVL, changed TOP field from /LONG to /FLOAT (Nov98)
changed behavior of UNDO LAST; renamed NEXT as HELP; renamed DONE
  as NEXT; added keyword RENORM and entry field PARS_NORM; changed
  MBAR_NEXT to MBAR_HELP (Dec98)
added output SCRATCH (Mar99)
bug correction re SCRATCH (FebMM)
```

B.111 LINEREM [/]

```
function        linerem
        remove lines from input spectrum and return the "cleaned" spectrum

syntax
        cleanspec=linerem(lamda,spec,sig=sig,cell=cell,nsigma=nsigma,$
        bkgval=bkgval,bkgerr=bkgerr,/quiet,/posve,/negve)

parameters
        lamda   [INPUT; required] wavelengths at which spectrum
                is defined.
        spec    [INPUT; optional] the spectrum.
                NOTE: if not given, LAMDA is taken to be SPEC and
                the array indices are taken to be LAMDA

keywords
        sig     [INPUT] error at each point; if not given, the
                errors are taken to be 1+sqrt(abs(SPEC)+0.75).
        nsigma  [INPUT; default: 4] multiple of SIG to consider
                as a threshold for detection of features
        cell    [INPUT] 1D filter to use in computing the background
                * default is [1,1,0,0,0,1,1]
                * if scalar, then [IC+1,ICC,IC+1], where
                  IC=intarr(2*CELL), ICC=intarr(2*CELL+1)
        bkgval  [OUTPUT] final background values at each bin
                NOTE: This is essentially a smoothed version of
                the cleaned spectrum!
        bkgerr  [OUTPUT] error estimates on BKGVAL
        quiet   [INPUT] if set, doesn’t show, doesn’t tell
        posve   [INPUT] if set, removes only +ve deviations
        negve   [INPUT] if set, removes only -ve deviations
        _extra  [JUNK] here only to prevent crashes

description
        1. make sure that the spectrum is defined on a regular grid.
           (if not, rebin [NOT IMPLEMENTED!])
        2. convolve the spectrum with background cell to determine
           local background.
        3. also propagate errors (assume gaussian; if poisson, use
           gaussian approximation)
        4. compare local value with local background
        5. flag those bins which are significantly different from local
           background (use +-NSIGMA*SIG as a threshold value)
        6. reset the flagged bin values to local background values
```

```
        7. repeat 2-6 until no new bins are flagged

these are all the smoothing tools in PINTofALE
        ALALOESS() makes a loess curve
        CLTSMOOTH() removes -ves from background subtracted spectra or light curves
        CONV_RMF convolves with specified RMF
        HAARTRAN() smoothes by filtering on Haar wavelet coefficients
        LINEREM() removes lines from spectrum by iterative S/N filtering
        NOISMOOTH() does boxcar accumulation a la Ebeling's asmooth
        REGROUP() accumulates from one end
        SCRMF does fast convolution using optimized RMF
        SMOOTHIE does peak-descent accumulation to build up S/N
        SPLAC computes a segmented piecewise linear approximations
        UNKINK() removes sharp kinks from a curve
        VARSMOOTH() does local smoothing at specified scales
        VOORSMOOTH() does fast local smoothing at specified scales

history
        vinay kashyap (Oct96)
        changed keyword SIGMA to SIG (VK; Jun98)
        CELL was being altered if input as scalar -- corrected; altered
          behavior of output when no deviations are found (VK; Oct98)
        now if QUIET>1, won't ask to quit if many bins are found (VK; 99Aug)
        changed POS and NEG to POSVE and NEGVE (VK; MMJul)
```

B.112 LINESPEC [/]

```
function        linespec
        returns a spectrum of atomic lines.  output will be in units
        of [erg,ph]/s/cm^2/[keV,Ang]

        if IKEV is set, WMIN,WMAX,WW must be input in [keV] and the
        output will have units .../keV

        if NOPH is set, output will have units of erg/...

        if EFFAR and WVLAR are not set, output will have units of .../cm^2

syntax
        spec=linespec(atom,wmin=wmin,wmax=wmax,nbin=nbin,/wlog,ww=ww,/ikev,$
        fstr=fstr,pres=pres,logP=logP,n_e=n_e,desig=desig,econf=econf,$
        /allah,dbdir=dbdir,dem=dem,abund=abund,effar=effar,wvlar=wvlar,$
        /noph,chifil=chifil,chidir=chidir,eqfile=eqfile,verbose=verbose)

parameters
        atom    [INPUT; default: all] element(s) whose line intensities are
                to be read.
                * can specify ionic state (e.g., 'FeXX')
                * may be an array (e.g., ['He', 'c 5', 'N V', 's_8'])

keywords
        wmin    [I/O] minimum value of wavelength range
                * default is 1.23985A (10 keV)
                * overwritten if WW is set
        wmax    [I/O] maximum value of wavelength range
                * default is 900A
                * overwritten if WW is set
        nbin    [I/O] number of bins in spectrum
                * default is 100
                * overwritten if WW is set
        wlog    [I/O] if set, binning will be logarithmic
        ww      [I/O] if set as an array on input, assumed to be the
                bin boundaries --
                * NBIN will be reset to n_elements(WW)-1 on output
                * WMIN will be reset to min([WW(0),WW(nbin)]) on output
                * WMAX will be reset to max([WW(0),WW(nbin)]) on output
        ikev    [INPUT] if set, WMIN,WMAX,WW are all assumed to be in keV
                * it is also caught at this stage and is not passed down to
                  LINEFLX
        fstr    [I/O] if defined on input, will return all the spectral
```

```
                lines info LINE_INT,LOGT,WVL,Z,ION,DESIG,ECONF in that order
                in a structure.
                * ion balance is included in LINE_INT, but abundances
                  are not.
                * if defined as a structure on input, RD_LINE and FOLD_IONEQ
                  will not be called, so use with caution!
        _extra  [INPUT] allows specifying defined keywords to subroutines
                called by this program
                * RD_LINE: PRES,LOGP,N_E,DESIG,ECONF,ALLAH,DBDIR,VERBOSE
                * FOLD_IONEQ: CHIFIL,VERBOSE
                * RD_IONEQ: CHIDIR,EQFILE
                * LINEFLX: DEM,ABUND,EFFAR,WVLAR,NOPH

subroutines
        RD_LINE [KILROY, SYMB2ZION [LAT2ARAB]]
        FOLD_IONEQ [WHEE, GETLOCMAX, RD_IONEQ [READ_IONEQ (CHIANTI)]]
        LINEFLX [WHEE]
        HASTOGRAM [KILROY]

history
        vinay kashyap (Jan97)
        included ion balance into FSTR (VK;Feb97)
        bug -- crashes when FSTR not specified (VK;Mar97)
        speeded up spectrum accumulation by adding call to HASTOGRAM (VK;Jan98)
        modified ion balance calcs (VK; 99May)
        changed keyword KEV to IKEV; DESIG and ECONF must be explicitly set
          to 0 to avoid reading them in in RD_LINE; streamlined IKEV behavior
          (VK; JanMMI)
```

B.113 LINESPEC_EM [/]

```
function        linespec_em
        returns a spectrum of atomic lines for a 1-T EM.  output will be in
        units of [erg,ph]/s/cm^2/[keV,Ang]

        if IKEV is set, WMIN,WMAX,WW must be input in [keV] and the
        output will have units .../keV

        if NOPH is set, output will have units of erg/...

        if EFFAR and WVLAR are not set, output will have units of .../cm^2

syntax
        spec=linespec_em(atom,wmin=wmin,wmax=wmax,nbin=nbin,/wlog,ww=ww,/ikev,$
        tlog=tlog,EM=EM,fstr=fstr,pres=pres,logP=logP,n_e=n_e,desig=desig,$
        econf=econf,/allah,dbdir=dbdir,abund=abund,effar=effar,wvlar=wvlar,
        /noph,chifil=chifil,chidir=chidir,eqfile=eqfile,verbose=verbose)

parameters
        atom    [INPUT; default: all] element(s) whose line intensities are
                to be read.
                * can specify ionic state (e.g., 'FeXX')
                * may be an array (e.g., ['He', 'c 5', 'N V', 's_8'])

keywords
        wmin    [I/O] minimum value of wavelength range
                * default on input is 1.23985A (10 keV)
        wmax    [I/O] maximum value of wavelength range
                * default on input is 900A
        nbin    [I/O] number of bins in spectrum
                * default on input is 100
        wlog    [I/O] if set, binning will be logarithmic
        ww      [I/O] if set as an array on input, assumed to be the
                bin boundaries --
                * NBIN will be set to n_elements(WW)-1 on output
                * WMIN will be set to min([WW(0),WW(nbin)]) on output
                * WMAX will be set to max([WW(0),WW(nbin)]) on output
        ikev    [INPUT] if set, WMIN,WMAX,WW are all assumed to be in keV
                * it is also caught at this stage and is not passed down to
                  LINEFLX
        fstr    [I/O] if defined on input, will return all the spectral
                lines info LINE_INT,LOGT,WVL,Z,ION,DESIG,ECONF in that order
                in a structure.
                * ion balance is included in LINE_INT, but abundances
```

```
                  are not.
                * if defined as a structure on input, RD_LINE and FOLD_IONEQ
                  will not be called, so use with caution!
        tlog    [INPUT] log10(T [K]) at which EM is defined
        EM      [INPUT] Emission Measure [cm^-5; if not, output units
                must be suitably mangled]
        _extra  [INPUT] allows specifying defined keywords to subroutines
                called by this program
                * RD_LINE: PRES, LOGP, N_E, DESIG, ECONF, ALLAH, DBDIR
                * FOLD_IONEQ: CHIFIL
                * RD_IONEQ: CHIDIR, EQFILE
                * LINEFLX: DEM, ABUND, EFFAR, WVLAR, NOPH

subroutines
        RD_LINE [KILROY, SYMB2ZION [LAT2ARAB]]
        FOLD_IONEQ [WHEE, GETLOCMAX, RD_IONEQ [READ_IONEQ (CHIANTI)]]
        LINEFLX [WHEE]
        HASTOGRAM [KILROY]

history
        vinay kashyap (Apr97; modified from LINESPEC.PRO)
        speeded up spectrum accumulation by adding call to HASTOGRAM;
          speeded up cycling thru lines by modifying DEM (VK;Jan98)
        modified ion balance calcs (VK; 99May)
        changed keyword KEV to IKEV; DESIG and ECONF must be explicitly set
          to 0 to avoid reading them in in RD_LINE; streamlined IKEV behavior
          (VK; JanMMI)
```

B.114 LMCOEFF [/fitting]

```
procedure       lmcoeff
        used by LEVMARQ (formerly MRQMIN.F) to evaluate the linearized
        fitting matrix ALPHA and vector BVEC (BETA in Numerical Recipes)
        and calculate chi^2

syntax
        lmcoeff,x,y,a,funcs=funcs,sig=sig,ifit=ifit,alpha=alpha,bvec=bvec,$
        chisq=chisq,yfunc=yfunc,poisson=poisson

parameters
        x       [INPUT; required] data points
        y       [INPUT; required] Y(X)
                * sizes of X and Y must match
        a       [INPUT; required] parameters for user-supplied procedure

keywords
        funcs   [INPUT] name of user-defined procedure that takes as input X
                and A, and returns YMODEL(X;A), and the partial derivatives
                of the model function wrt A.  Any procedure that was written
                to work with CURVEFIT or GHRS' WFIT will do.
                * default is set to X3MODEL
                * why is it a procedure and not a function?
                  well, ask the person who wrote CURVEFIT.
        sig     [INPUT; default=sqrt(abs(Y)+0.75)+1] standard deviations on Y
        ifit    [INPUT] integer array of same size as A, with 0's indicating
                frozen parameters and 1's indicating thawed parameters
                * if IFIT does not match A for any reason, all parameters
                are assumed to be thawed
        alpha   [OUTPUT] fitting matrix
        bvec    [OUTPUT] fitting vector
        chisq   [OUTPUT] chi^2, or ln(p(D|M)) if POISSON is set
        yfunc   [OUTPUT] Y(X;A) -- function values
        poisson [INPUT] if set, returns the log of the poisson likelihood,
                p(D|M) instead of the chi-sq
                * do _not_ use this keyword for background-subtracted data!!
        _extra  [JUNK] here only to prevent crashing the program

subroutines
        LNPOISSON

history
        (C) Copr. 1986-92 Numerical Recipes Software =$j!]Y'1,).
        translated from MRQCOF.F to IDL by Vinay Kashyap (Oct98)
```

```
now returns correct chisq if all params are frozen (VK; JanMM)
added keyword POISSON and call to LNPOISSON (VK; Aug01)
```

B.115 LNPOISSON [/stat]

```
function        lnpoisson
        returns the natural log of the poisson likelihood of observing
        the specified counts given the model source intensity,
                p(D|M) = M^D * exp(-M) / D!

syntax
        lnp=lnpoisson(d,m,/chilike,verbose=verbose)

parameters
        d       [INPUT; required] data counts
        m       [INPUT; required] model intensity
                * output will be array of size [N(D),N(M)]
                  unless CHILIKE is set, in which case output will be N(M)
                  (beware that IDL automatically compresses trailing
                  dimensions if any of them are equal to 1)

keywords
        chilike [INPUT] if set, returns -2*SUM_{D}(ln(p(D|M)))
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to avoid crashing the program

history
        vinay kashyap (Aug01)
        bug correction: m^d was not translated correctly (VK; Mar02)
```

B.116 LOCMAX [/util]

```
function        locmax
                returns the local maxima of a 1D array

syntax
        fmax=locmax(f,sigma=sigma,thresh=thresh,width=width,prob=prob,$
                /poisson,nubin=nubin,fmin=fmin,fmax=fmax)

parameters      f       [INPUT; required] input array

keywords        sigma   [INPUT] errors on F (ignored if POISSON is set)
                        * ignored if POISSON is set AND F(*).GE.0
                        * default is sqrt(abs(F)+0.75)+1
                thresh  [INPUT; default: 0.5] threshold probability below
                        which to ignore a calculation
                width   [INPUT; default: 1] compare value in bin to values
                        on either side averaged over WIDTH bins
                        * NOTE: WIDTH is the *half*width
                poisson [INPUT] if set, assumes a poisson error distribution
                        * automatically UNSET if min(F) < 0
                prob    [OUTPUT] returns the computed probabilities

        the following are not to be changed unless the user understands
        what's going on!

                nubin   [INPUT; default: 100] size of uniform grid to
                        superpose on F
                fmin    [INPUT] minimum value for integration.
                        default is 0.2*min(F) or if min(F)<0 then 5*min(F)
                fmax    [INPUT] maximum value for integration.
                        default is 5*max(F) or if max(F)<0 then 0.2*max(F)

description
        at each point, compute the probability that given value is a
        local maximum.
          p = \int dF p(f|f[i],s[i])*p(f[i-1]<f|s[i-1])*p(f[i+1]<f|s[i+1])
        find all the peaks in the distribution of probabilities that
        lie above a given threshold.  the positions of these peaks are
        identified with positions of local maxima.

requires
        LOCMAX_CDF (included in file)
        IGAMMA (not in old IDLs)
        ERRORF (not in old IDLs?)
```

```
history
	vinay kashyap (Oct96)
```

B.117 LOCUS_ELLIPSE [/util]

```
procedure       locus_ellipse
        make a curve describing the locus of an ellipse

syntax
        locus_ellipse,xout,yout,xcen=xcen,ycen=ycen,amajor=amajor,$
        aminor=aminor,theta=theta,npt=npt,verbose=verbose

parameters
        xout    [OUTPUT; required] x-coordinates of the locus
        yout    [OUTPUT; required] y-coordinates of the locus

keywords
        xcen    [INPUT; default=0] x-coordinate of the center
        ycen    [INPUT; default=0] y-coordinate of the center
        amajor  [INPUT; default=1] length of the semi-major axis
        aminor  [INPUT; default=1] length of the semi-minor axis
                * if AMINOR is not defined, it is set to AMAJOR
                  to make a circle
                * if AMINOR>AMAJOR, no harm done, program will
                  run regardless of mathematical purity
        theta   [INPUT; default=0 degrees] anticlockwise angle made
                by AMAJOR with x-axis
        npt     [INPUT; default=101] number of points along the locus,
                spaced equally along THETA
                * note that the first and the last points are identical
                  unless NPT < 0, in which case abs(NPT) points are
                  returned without the overlap
        verbose [INPUT; default=0] controls chatter

        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (Mar2008)
```

B.118 LOGIT [/stat]

```
function        logit
        compute and return the logit function.

        LOGIT() maps a variable in the [0,1] range to the
        real number line [-\infty,+\infty] by applying the
        transformation ln(x/(1-x)).
        conversely, UNLOGIT() maps the real number line to
        [0,1] by applying the transformation e^y/(1+e^y).

syntax
        lx=logit(x,sigx,sigl=sigl)

parameters
        x       [INPUT; required] a variable in the range [0,1]
                * can be an array
                * value is set to NaN in output if input falls
                  outside valid range
        sigx    [INPUT] standard error on X
                * if <0 and >-1, then ABS(SIGX) is taken to be the
                  fractional error
                * if <-1 and >-100, then ABS(SIGX) is taken to be
                  the percentage error
                * if <-100, then 1/ABS(SIGX) is taken to be the
                  fractional error

keywords
        sigl    [OUTPUT] standard error on LOGIT(X)
                * note that SIGL = SIGX/(X*(1-X))
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (3may04; based on conversation with Xiao-li Meng)
```

B.119 LOOPEM [/]

```
function        loopem
        convert a set of integrated emission measures to a differential
        emission measure distribution assuming a hydrostatic-loop type
        distribution and return computed DEM

        DEM(T) is generally n^2*dX/dlnT, and
        EM(T) == \int_0^T dlnT' DEM(T') = \int_0^T dT' DEM(T')/T'
        In particular, DEM is parameterized as DEM(T)=a*T^b
        So for a given EM(T), we obtain
                EM(T)=(a/b)*(Tmax^(b)-Tmin^(b)), Tmin=0, b>0
        with Tmin=0 for b>0

syntax
        dem=loopem(logTmax,EM,logT=logT,sloop=sloop,verbose=verbose)

parameters
        logTmax [INPUT; required] loop top temperature(s) in log10(deg K)
        EM      [INPUT] if given, normalize the output DEM to give a
                total Emission Measure equal to this.
                * size must match that of logTmax -- if fewer, fills out
                  with 1st element; if more, ignored

keywords
        logT    [INPUT] temperature grid over which output DEM is returned
                * default is 4..8 in steps of 0.05
        sloop   [INPUT] slope of DEM in log(DEM)-log(T) space
                * default is 1.5
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (Nov01)
        changed NORM calc to allow for large values of SLOOP (VK; Feb05)
```

B.120 LOSRAD [/esempio]

```
script  losrad
        compute radiative loss function

syntax
        .run losrad

warning
        all internal program variables have names that begin as "v_" or "jnk_"
        if you have any preexisiting variables with these names, they will be
        overwritten.

inputs
        !LDBDIR  line emissivity database
        !CDBDIR  continuum emissivity database directory
        !CEROOT  continuum emissivity database root
        !IONEQF  ion-balance file
        !EDENS   electron density
        !ABUND   abundances
        !VERBOSE controls chatter
        V_WMIN   [1.0 Ang] minimum wavelength to consider
        V_WMAX   [3000.0 Ang] maximum wavelength to consider

outputs
        V_LLOGT  temperatures at which radiative loss is computed
        V_LOSS   radiative loss function P(T)
        V_LLOS   loss function due to lines
        V_CLOS   loss function due to continuum processes
        V_LLOSE  loss function due to excitation transitions only
        V_LLOSR  loss function due to radiative transitions only
        V_LLOSI  loss function due to inner-shell ioniszation transitions only

how to use
        1. initialize the PINTofALE environment using INITALE
        2. the parameters that will be set with INITALE are:
           --   ATOMIC: !LDBDIR, !IONEQF, !CDBDIR, !CEROOT, !CHIDIR
           --   STELLAR: !EDENS, !ABUND, !NH, !FH2, !HE1, !HEII
           verify that they are initialized to the right values
        3. set the wavelength range of interest
                V_WMIN = 1.0
                V_WMAX = 3000.
        4. run this script

history
```

```
vinay kashyap (May99; modified from radloss.pro)
major cosmetic overhaul (VK; DecMM)
minor cosmetic changes (VK; Jul01)
changed default WMAX to 3000 AA (VK; Sep01)
plot now handles the 24-bit color case (VK; Apr03)
updated for IDL5.6 keyword_set([0]) behavior change for vectors
  (VK; 20Mar2006)
allowed for APED emissivities (VK; 6Aug2009)
```

B.121 LSD [/]

```
function        lsd
        returns list of density sensitive lines in given wavelength range

syntax
        ld=lsd(wrange,fluxes,wvls,elem=elem,edens=edens,DEM=DEM,tlog=tlog,$
        dbdir=dbdir,ceiling=ceiling,flor=flor,ratmax=ratmax,flxmax=flxmax,$
        outZ=outZ,outIon=outIon,outIDX=outIDX, tol=tol,chifil=chifil,$
        chidir=chidir,eqfile=eqfile,/temp,abund=abund,/noph,effar=effar,$
        wvlar=wvlar,/ikev)

parameters
        wrange  [INPUT; required] wavelength range in which to search for
                density sensitive lines
                * if only one element is given, set the range to be 10% of
                  it on either side
                * if the single element is a string, apply RD_LIST rules
                  to decipher it ("WVL +- dW", "WMIN,WMAX", "WMIN-WMAX"
                  are all allowed)
        fluxes  [OUTPUT] 2D array of predicted fluxes for lines that are
                found to be density sensitive, FLUXES(EDENS,WVLS)
        wvls    [OUTPUT] array of wavelengths of lines that are found to
                be density sensitive

keywords
        elem    [INPUT] if set, confine attention to just those elements and
                ionic species specified.
                * default is to look at all available lines
        edens   [I/O] array of electron densities [cm^-3] at which to
                check for density sensitivity.
                * default is [1e8,1e14]
                * if not specified, insufficiently specified, or otherwise
                  meaningless, uses default
        DEM     [INPUT] input array of differential emission measures [cm^-5]
                * default is to use 1e12 [cm^-5] at specified TLOG
        tlog    [INPUT] log(temperatures) at which DEM is defined
                * if not specified, set to 6.0, unless DEM has more steps,
                  in which case interpolated to the 4..8 range
        dbdir   [INPUT] line database directory to use to search for the
                density sensitive lines
                * default is $CHIANTI
        ceiling [INPUT] maximum relative variation the fluxes of a given
                line must have before it is accepted as density sensitive
                * default is 2
```

```
                * if 0 < CEILING < 1, it is assumed that the *reciprocal*
                  of CEILING is given (e.g., CEILING=0.5 is translated to 2)
                * if CEILING < 0, assumed to be percentages (e.g., CEILING=-1
                  is translated to 1/0.01 == x100)
        flor    [INPUT] if set, ignores any line which has a peak flux
                a factor FLOR below the flux of the strongest of the
                lines filtered through CEILING.
                * default is 0
                * if FLOR > 1, assumed that the reciprocal of FLOR is given,
                  i.e., FLOR=100 translates to 0.01
                * if FLOR < 0, taken to be the absolute flux threshold, in
                  whatever units it is being calculated in
                * NOTE on difference between CEILING and FLOR:-
                  CEILING works on ratios, and FLOR works on the intensities
        ratmax  [OUTPUT] the maximum deviations found for each of the
                density-sensitive lines found
        flxmax  [OUTPUT] the peak fluxes for each of the density sensitive
                line found
        outZ    [OUTPUT] atomic numbers
        outIon  [OUTPUT] atomic numbers
        outIDX  [OUTPUT] indices which passed the test
                * be careful in how you use this.  a common mistake is to
                  assume that it matches with whatever line emissivity
                  database that has been read in.  that might very well be
                  true, but is definitely not guaranteed.

        _extra  [INPUT] use this to pass defined keywords to subroutines:-
                FOLD_IONEQ: CHIFIL
                RD_IONEQ: CHIDIR, EQFILE
                LINEFLX: TEMP, ABUND, NOPH, EFFAR, WVLAR, IKEV
                ARRAYEQ: TOL

example usage
        ls=lsd(’10-120’,dbdir=’$CHIANTI’,edens=[1e9,1e12],ceiling=2,ratmax=rmx)
        for i=0,n_elements(ls)-1 do print,ls(i),1./rmx(i),i
        ;
        eden=10.^(findgen(6)+8)
        ls=lsd(’113.35?0.1’,flx,elem=’Fe20’,dbdir=’$CHIANTI’,edens=eden)
        plot,eden,flx

subroutines
        MK_DEM
        RD_LINE
        FOLD_IONEQ
            RD_IONEQ
```

```
                READ_IONEQ (CHIANTI subroutine)
        LINEFLX
        ARRAYEQ
        KABOOM
        INICON
        IS_KEYWORD_SET

history
        vinay kashyap (Nov98)
        changed default EDENS to [1e8,1e14]; added keywords DEM,TLOG,OUTZ,
          OUTION; added call to MK_DEM; changed keyword FLOOR to FLOR; changed
          behavior of FLOR<0 (VK; Dec98)
        modified ion balance calcs (VK; 99May)
        changed call to INITSTUFF to INICON (VK; 99May)
        added keyword outIDX (VK; JanMMI)
        updated for IDL5.6 keyword_set([0]) behavior change for vectors
          (VK; 20Mar2006)
```

B.122 MAKE_LOWRES [/esempio]

```
procedure     make_lowres
        wrapper script to run hires2lowres and put the simulated
        ACIS spectral data into a ASCII tables and fits files

syntax
        make_lowres,hphafile,mphafile,hgarffile,mgarffile,$
        hhetgwvl,hhetgcts,mhetgwvl,mhetgcts,hetgwvl,hhetgflx,$
        hhetgflxerr,mhetgflx,mhetgflxerr,tothetgflx,tothetgflxerr,$
        aciswvl,aciscts,aciserr,hphahdr,mphahdr,hgarfhdr,mgarfhdr,$
        acisarffile,acisrmffile,debug=debug,txtfile=txtfile,$
        fitsfile=fitsfile,flxfits=flxfits,flxtxt=flxtxt

parameters
        hphafile     [INPUT; required] PHA type I spectral file
                     containing the HEG HETG spectrum for a given
                     order
        mphafile     [INPUT; required] PHA type I spectral file
                     containing the MEG HETG spectrum for a given
                     order
        hgarffile    [INPUT; required] Gratings ARF corresponding to
                     the HEG PHA file
        mgarffile    [INPUT; required] Gratings ARF corresponding to
                     the MEG PHA file
        hhetgwvl     [OUTPUT; required] The midpoints of the HEG
                     wavelength bins
                     * units are Angstroms
        hhetgcts     [OUTPUT; required] The background-subtracted
                     counts spectrum extracted from the HEG PHA file
                     * units are [cts/s/AA]
        mhetgwvl     [OUTPUT; required] The midpoints of the MEG
                     wavelength bins
                     * units are Angstroms
        mhetgcts     [OUTPUT; required] The background-subtracted
                     counts spectrum extracted from the MEG PHA file
                     * units are [cts/s/AA]
        hetgwvl      [OUTPUT; required] The midpoints of the common
                     HEG, MEG, and HEG+MEG flux spectrum's
                     wavelength grid
                     * grid extends ~0-30 AA at HEG resolution
        hhetgflx     [OUTPUT; required] Adaptively-smoothed,
                     background-subtracted HEG flux spectrum
                     * units are [ph/s/cm^2/AA]
        hhetgflxerr  [OUTPUT; required] The error on the HEG flux
```

```
                spectrum
mhetgflx        [OUTPUT; required] Adaptively-smoothed,
                background-subtracted MEG flux spectrum
                * units are [ph/s/cm^2/AA]
mhetgflxerr     [OUTPUT; required] The error on the MEG flux
                spectrum
tothetgflx      [OUTPUT; required] Adaptively-smoothed,
                background-subtracted coadded HEG+MEG flux
                spectrum
                * units are [ph/s/cm^2/AA]
tothetgflxerr [OUTPUT; required] The error on the coadded
                HEG+MEG flux spectrum
aciswvl         [OUTPUT; required] The midpoints of the
                predicted ACIS spectrum's wavelength grid
aciscts         [OUTPUT; required] The predicted ACIS spectrum
                * units are [cts/s/AA]
aciserr         [OUTPUT; required] The error on the predicted
                ACIS spectrum
hphahdr         [OUTPUT; required] The header of the HEG PHA
                file
mphahdr         [OUTPUT; required] The header of the MEG PHA
                file
hgarfhdr        [OUTPUT; required] The header of the HEG gARF
                file
mgarfhdr        [OUTPUT; required] The header of the MEG gARF
                file
acisarffile     [INPUT; required] The ACIS ARF
acisrmffile     [INPUT; required] The ACIS RMF

keywords
        debug           [INPUT] controls debugging chatter
        txtfile         [INPUT; required] Output text file for the HETG
                        and predicted ACIS spectra
        fitsfile        [INPUT; required] Output FITS file for the HETG
                        and predicted ACIS spectra
        flxfits         [INPUT; required] Output FITS file for the HEG,
                        MEG, and combined HEG+MEG flux spectra
        flxtxt          [INPUT; required] Output text file for the HEG,
                        MEG, and combined HEG+MEG flux spectra

restrictions
        requires conv_rmf, smoothie, and rebinw from PINTofALE,
        available at http://hea-www.harvard.edu/PINTofALE/

history
```

`Owen Westbrook, Peter Mendygral, and John Slavin (Mar07)`

B.123 MAKE_SPECTRUM [/scrypt]

```
script  make_spectrum
        compute a spectrum using line and continuum emissivities

warning
        all program variables have names that begin as "v_" or "jnk_"
        if you have any preexisiting variables with these names, they
        will be overwritten.

inputs
        ATOMIC:
          line emissivities and wavelengths
          continuum emissivities and wavelength grid
          (subsumed in database directory names)
          ion fractions
        STELLAR:
          coronal electron density
          coronal abundances
          Differential Emission Measure
          distance
          Galactic coordinates (or absorption column density)
        OBSERVATIONAL:
          instrument effective area
          synthetic spectrum wavelength grid
          photon redistribution matrix
          exposure time

outputs
        v_linint        line intensities @ source
                (@ wavelengths v_lwvl; from species v_Z,v_ion)
        v_conint        continuum intensities @ source
                (@ wavelength grid v_cww)
        v_linflx        theoretical line fluxes @ earth
                (using ISM transmission factors v_ltrans)
        v_conflx        theoretical continuum fluxes @ earth
                (using ISM transmission factors v_ctrans)
        v_linspc        theoretical line spectrum @ earth
        v_conspc        theoretical continuum spectrum @ earth
        v_flxspc        predicted spectrum through telescope
        v_wvls          mid-bin values of predicted spectrum wavelength grid
        v_ctspc         predicted count spectrum for detector
        v_chan          energy channels for count spectrum

how to use
```

```
        1. initialize the PINTofALE environment using INITALE
        2. the parameters that will be set with INITALE are:
           --   ATOMIC: !LDBDIR, !IONEQF, !CDBDIR, !CEROOT, !CHIDIR
           --   STELLAR: !EDENS, !ABUND, !NH, !FH2, !HE1, !HEII
           verify that they are initialized to the right values
        3. define DEM as a function of LOGT (use DEMACS() or MK_DEM())
           --   place in variables v_LOGT [K] and v_DEM [cm^-3]
        4. define distance in [pc]
           --   place in variable v_DIST
        5. read in the instrument effective area
           --   place in variables v_EFFAR [cm^2] and v_WVLAR [Ang]
        6. read in (or define) the synthetic spectrum wavelength grid
           --   place in variable v_WGRID [Ang]
           (must be _bin boundaries_)
        7. define the exposure time [ks]
           --   place in variable v_EXPTIME
        8. define the name of the OGIP-style RMF
           --   place in variable v_RMF
           (set to 'NONE' if, e.g., grating spectrum is being analyzed)
        9. run this script

history
        vinay kashyap (OctMM)
```

B.124 MARFRMF [/util]

```
function        marfrmf
        returns the instrument response matrix computed as the product of
        the redistribution matrix with the ancillary response.

syntax
        rsp=marfrmf(arffil,rmffil,ostr,effar=effar,/noext,/nofull,$
        fchcol=fchcol,/shift1)

parameters
        arffil  [INPUT; required] OGIP-compliant Ancillary Response File
                * appends a ".arf" if missing (overridden by NOEXT)
        rmffil  [INPUT; required] OGIP-compliant Redistribution Matrix File
                * appends a ".rmf" if missing (overridden by NOEXT)
        ostr    [OUTPUT] structure containing the axes information for the
                response matrix -- see RD_OGIP_RMF()

keywords
        effar   [OUTPUT] returns the effective area as a function of ELO
        noext   [INPUT] normally, if an extension is missing from ARFFIL
                or RMFFIL, the program will append the appropriate extension.
                if this keyword is set, no such action will occur.
        nofull  [INPUT] if set, bypasses RDRESP() and the accompanying
                computation of the full response matrix (i.e., with the OGIP
                compression removed).  this is useful for those cases where
                the uncompressed matrix is too large for memory.
                * if this is set, the output will be simply the same as
                  OSTR.MATRIX, which is the compressed version of the
                  effective area weighted redistribution matrix
        _extra  [INPUT] pass defined keywords to subroutines
                RD_OGIP_RMF: FCHCOL, SHIFT1

restrictions
        * requires subroutines RD_OGIP_RMF, RDRESP and RDARF
        * requires IDLASTRO library of routines

history
        vinay kashyap (Jul97)
        allowed for keyword SHIFT1 (VK; JanMMI)
        added keyword NOFULL; now multiplication also works on OSTR.MATRIX
          (VK; Nov2001)
        changed output in case of /NOFULL from 1.0 to OSTR.MATRIX (VK; Oct2008)
```

B.125 MC_EROR [/util]

```
procedure mc_eror

          simple procedure that calculates confidence bounds
          about a given parameter value g given a certain number
          of monte carlo simulations contained in array sims.
          assumes the simulations are distributed normally about
          the 'candidate' parameters.

          by default,set to find smallest possible bounds. can also
          be set to find confidence bounds by counting  upward
          and downwards by required fraction.

syntax
           mc_eror, g, sims, erru, errl, confidence=confidence

parameters

         g     [input, required] parameter about which to estimate bounds
         sims  [input, required] array of monte carlo simulated g's
                                 minimum 10 simulations [though this
                                 seems a ridiculous number]
         bndu  [output]          upper confidence bound
         bndl  [output]          lower confidence bound

keywords

         confidence [default = 0.68269] confidence limit at which bounds
                                    are desired.
         alas                    find confidence bounds by counting  upward
                                 and downwards by required
                                 fraction. may also be set
                                 to find smallest possible bounds.

history
         LL Jan03
```

B.126 MC_EROR [/util]

```
procedure mc_eror

          simple procedure that calculates confidence bounds
          about a given parameter value g given a certain number
          of monte carlo simulations contained in array sims.
          assumes the simulations are distributed normally about
          the 'candidate' parameters.

          by default,set to find smallest possible bounds. can also
          be set to find confidence bounds by counting  upward
          and downwards by required fraction.

syntax
           mc_eror, g, sims, erru, errl, confidence=confidence

parameters

         g     [input, required] parameter about which to estimate bounds
         sims  [input, required] array of monte carlo simulated g's
                                 minimum 10 simulations [though this
                                 seems a ridiculous number]
         bndu  [output]          upper confidence bound
         bndl  [output]          lower confidence bound

keywords

         confidence [default = 0.68269] confidence limit at which bounds
                                    are desired.
         alas                    find confidence bounds by counting  upward
                                 and downwards by required
                                 fraction. may also be set
                                 to find smallest possible bounds.

history
         LL Jan03
```

B.127 MCERROR [/fitting]

```
procedure       mcerror
        procedure that estimates confidence bounds for model fits by
        fitting to different monte-carlo realizations of the data and
        estimating the bounds from the distribution of the resulting
        parameter values.

        generates NN number of synthetic data sets by randomly varying
        each data point within its estimated error. fits these NN
        synthetic data  sets to generate parameter sets a(1),a(2),..a(NN),
        so that we can estimate distribution of a(x)-a(best).  Uses this
        as a basis for confidence bound estimation.  Finds smallest possible
        bounds giving required confidence (as opposed to 'counting' up
        and down from best fit by required fraction)

syntax
        mcerror,x,y,a,erru,errl,ysig=ysig,freeze=freeze,mcfuncs=mpfuncs,$
        algo=algo,verbose=verbose, erra=erra,yfunc=yfunc,mciter=mciter,$
        itmax,chitr=chitr,mcconf=mcconf,/dumb, jumpup=jumpup,jumpdn=jumpdn,$
        =svdthr,funcs=funcs, function_name=function_name,$
        ype, missing=missing,/fwhm,/norm,betap=betap, /poisson

parameters
        x       [INPUT; required] data points
        y       [INPUT; required] Y(X)
                * sizes of X and Y must match
        a       [I/O; required] parameters for user-supplied function
                * on input, these are assumed to be initial guesses
                * on output, these contain the best-fit values
        erru    [OUTPUT; required] upper limits of confidence range interval
        errl    [OUTPUT] lower limits of confidence range interval
                * if ERRL is not specified on input, ERRU will contain the
                  average value of the upper and lower >>deviations<<.

keywords
        ysig    [INPUT] standard deviations on Y
                * default=sqrt(abs(Y)+0.75)+1
                * if single element, then sig(Y[*])=YSIG(0)
                * if -ve, taken to be the fractional error
       mciter  [INPUT] number of additional iterations for each parameter:
               total iterations = (mciter x number of parameters) > 2xmciter+1
               default: mciter=30
       mcconf  [INPUT] (two-sided) confidence for bound estimation,
                       default: mpfunc =0.68
```

```
       mcsims  [OUTPUT] 2 dimensional array MCSIMS==MCSIMS(free,sims)
               where free are the number of thawed paramters and sims
               is the number of simulations. contains all simulations
               on output
       mpfunc  [INPUT] contains name of input function to fit
                       REQUIRED WHEN USING MPFIT
        freeze  [INPUT] freeze numbered parameters (index starts from 0!)
        algo    [INPUT] fitting algorithm
                * only the following are implemented:
                -- LevMarq+SVD
                -- IDL-Curvefit
               -- Mardkwardt's MPFIT
        verbose [INPUT] verbosity level
        erra    [OUTPUT] formal "curvature" errors on the best-fit parameters
        yfunc   [OUTPUT] best-fit Y(X;A)
        _extra  [INPUT] pass defined variables to subroutines:-
                FIT_LEVMAR: ITMAX, CHITHR, DUMB
                ADJUSTIE: TIES, VNAME
                LEVMARQ: JUMPUP, JUMPDN, SVDTHR
                LMCOEFF: FUNCS, POISSON
                CURVE_FIT: FUNCTION_NAME
                note:-  FUNCS, FUNCTION_NAME, and MPFUNC refer to name
                        of user-defined function that takes as input X
                        and A, and returns YMODEL(X;A), and the partial
                        derivatives of the model function wrt A.  Any
                        function that was written to work with CURVEFIT
                        or GHRS' WFIT will do.
                        The default for FIT_LEVMAR is X3MODEL.
                MK_3MODEL: TYPE
                MK_GAUSS: MISSING, FWHM, NORM
                MK_LORENTZ: BETAP, MISSING, NORM

history
        Liwei Lin (Apr03) idenentical to ERORS.pro in input/output/parameter checks
          added keyword MPFUNC for compatability with MPFIT
          added keyword MCCONF so user can toggle confidence required
                 BUG FIX, mcerror not mcerors LL/SC JUL03
                 BUG FIX, yfunc was being recursivly overwritten in
                          synthetic loop LL/SC JUL03
          added forward_function (VK; Jul03)
        some cosmetic changes (VK; Feb04)
         added MCSIMS (LL; Mar04)
       BUG FIX algo keyword incorrectly processed (LL ; Sep05)
```

B.128 MCMC_ABUND [/fitting]

```
function    mcmc_abund

        An abundance-independent DEM-reconstruction can be achieved
        by running a Monte Carlo Markov-Chain DEM reconstruction
        mcmc_dem() on a set of He-like to H-like flux ratios.

        A DEM realized in this fashion together with the observed
        line fluxes give implied abundances. This function returns
        fractional abundnaces relative to solar 'grevesse & sauval'
        (not logarithmic values) given a set of MCMC realized DEMS
        (mcmc_dem ouput), fluxes, and line details.

parameters
        simdem [INPUT;required]  an array of size
                (NT,NSIM+1), with last column containing the best-fit
         logt   [INPUT] log_10(Temperature [K]) at which DEMS are given
        wvl     [INPUT; required] wavelengths [Ang]
         flx    [INPUT; required] flux [ph/s] at each WVL ([ph/s/cm^2]
        fsigma [INPUT; required] errors for flx
         emis   [INPUT; required] line emissivities [1e-23 ergs cm^3/s]
                * EMIS==EMIS(LOGT,WVL)
        z      [INPUT; required] atomic numbers of the elements
               generating the   lines at WVL.
        asig   [OUTPUT] error in the weighted mean abundance
        zout   [OUTPUT] atomic numbers corresponding to output abundances
keywords
        abund  [INPUT] set  of abundances. see GETABUND(). Output
               will be relative to this set. Default is Grevesse &
               Sauval (1998)
        abweight [INPUT] if a different weighting is required to compute
               abundances then specify abweight as an array of weights of
               the same size as flx (e.g. one can use fsigma)
        elogt  [INPUT] log_10(Temperature[K]) at which EMIS are
                       defined if different from logt
        nosol  [INPUT] if set, then output abundances will not be
                       relative to solar
        olog   [INPUT] if set, outputs abundances and asig will be
               logarithmic values.
        crctn_factor [INPUT] if set, correctin factor is applied to
                     theoretical fluxes before calculation of abundnaces
        _extra [INPUT ONLY] use this to pass defined keywords to subroutines

subroutines
```

```
        MC_EROR (/home/llin/mc_eror.pro) not PoA standard yet
        LINEFLX
history
        Liwei Lin 5/03
           ADDED keyword log for logarithmic output (LL 7/03)
           BUG FIX elogt is actually taken into account when set (LL/WB 8/03)
        LL/CA 9/03 BUGFIX Costanza points out error calculation bug
           which gives overestimation in errors
       BUG FIX elogt handler (call to rebinx) should have logt
           not dlogt as input (LL/WB 9/03)
        LL 01/04
           ADDED keyword crctn_factor for use with mixie
        LL 01/04
           H abundance cannot be updated
        LL/SC 2/04
           BUGFIX HISTOGRAM will give a one element array if given
           just one atomic number to bin up.
        LL ADDED _extra keyword to lineflx
        LL 5/05 Change the asig determination to only include the errors
           as determined from the DEMs.THIS SHOULD HAVE BEEN FIXED 9/03 AS
           stated above. (do not include the error in the
           fluxes a second time).
           ADDED keyword abweight
        LL 6/05 ADDED keyword abund
           BUGFIX previous error calculation would fail for highly
           assymetric cases
        LL/DG 6/05 BUGFIX crash when MC_EROR returns two-element
           arrays
```

B.129 MCMC_DEM [/fitting]

```
function        mcmc_dem
        performs Markov-Chain Monte-Carlo using a Metropolis algorithm
        on a set of supplied line fluxes and returns an estimate of the
        DEM that generates observed fluxes.

syntax
        dem=mcmc_dem(wavelengths,fluxes,emissivities,Z=Z,logt=logt,$
        diffem=initial_DEM,abund=abundances,fsigma=flux_errors,$
        onlyrat=onlyrat,sysdev=sysdev,sysgrp=sysgrp,$
        sampenv=sampenv,storpar=storpar,storidx=storidx,$
        stordem=stordem,bestpar=bestpar,$
        simprb=simprb,simdem=simdem,simabn=simabn,simflx=simflx,simprd=simprd,$
        ulim=upper_limit_flag,softlim=softlim,demrng=DEMrange,abrng=abrng,$
        smoot=smooth,loopy=loopy,/spliny,verbose=verbose, xdem=DEM_bins,xab=ABUND_bins,$
        nsim=nsim,nbatch=nbatch,/steps,weight=weight,/gauss,type=type,/boxcar,$
        nburn=nburn,/nosrch,savfil=savfil,bound=bound,demerr=demerr,$
        aberr=aberr,/temp,/noph,effar=effar,wvlar=wvlar,/ikev,/regrid,$
        mproxy=mproxy,mplet=mplet)

parameters
        wvl     [INPUT; required] wavelengths [Ang] at which fluxes are
                observed
                * NOTE: these are used only if the keyword NOPH is _not_ set,
                  i.e., when the input is to be converted from [erg/...] to
                  [ph/...].  In cases when the EMIS is precomputed with the
                  effective area and abundances (e.g., for filter data from
                  solar observatories), this is a dummy variable.
        flx     [INPUT; required] flux [ph/s] at each WVL ([ph/s/cm^2]
                if EFFAR and WVLAR are not passed to LINEFLX)
        emis    [INPUT; required] line emissivities [1e-23 ergs cm^3/s]
                * EMIS==EMIS(LOGT,WVL)
                * DEM is computed at all temperatures that EMIS are defined.
                * NOTE: the units are assumed to be [1e-23 ergs cm^3/s],
                  but of course the code doesn't care what they are exactly.
                  So it is perfectly possible to have them in units of, e.g.,
                  [DN cm^5 s^-1 pix^-1] if one so chooses (see SOLAR_TRESP.PRO)

keywords
        Z       [INPUT] atomic numbers of the elements generating the
                lines at WVL.
                * if not given, assumed to be Fe
                * if EMIS are precalculated including abundances, then
                  set this to *1* (to mimic H)
```

```
logt     [INPUT] log_10(Temperature [K]) at which EMIS are given
         * if not supplied, assumed to go from 4 to 8
         * if there is a size mismatch, logT will get stretched/shrunk
           to cover EMIS
diffem   [INPUT] initial guess for DEM(LOGT); default is 1e14 [cm^-5]
abund    [I/O] abundances.  default is to use Anders & Grevesse (see
         GETABUND)
         * if ABRNG is non-trivial, will be allowed to vary and on
           exit will contain the MAP estimates of the abundances.
         * minimum is hardcoded at 1e-20
fsigma   [INPUT] error on FLX; if not given, taken to be
         1+sqrt(abs(FLX)+0.75)
ulim     [INPUT] long-integer array specifying which of the fluxes
         are upper limits (1: UL, 0: not)
onlyrat  [INPUT] string array describing which of the input fluxes
         must be considered only as ratios
         * each element of ONLYRAT corresponds to the fate of the
           corresponding element of FLX
         * basic format is: "sP#[,sP#[,...]]" where
           -- "#" is an integer flag describing the ratio being
              constructed
           -- "P" is a positional descriptor and can take on
              values N (for numerator) or D (for denominator)
           -- "s" stands for the sign with which the flux
              goes into the ratio "+" or "-"
         * e.g., to construct a simple ratio FLUXES[2]/FLUXES[1],
           ONLYRAT=['','+D1','+N1']
         * e.g., to construct two hardness ratios
                 FLUXES[3]/FLUXES[1] and
                 (FLUXES[3]-FLUXES[1])/(FLUXES[3]+FLUXES[1]),
           ONLYRAT=['','+D1,-N2,+D2','','+N1,+N2,+D2']
sysdev   [INPUT] systematic deviations in the underlying atomic
         physics data, multiplies the emissivity for a given line
         by a random number in each iteration to mimic systematic
         uncertainty in EMIS
         * it is assumed that these modifications are small -- if
           you do use this option, please check the output traces
           to make sure that the chain is stable
         * should be an array of the same size as FLX, and each
           number defines the range over which EMIS[*,WVL] varies,
           10.^(randomn(seed)*SYSDEV[i])*EMIS[*,i]
         * if size is smaller than FLX, the missing elements are
           set to 0, i.e., no modifications are made
sysgrp   [INPUT] group different rows in EMIS to vary the same way
         in each iteration, is simply an array of 1-based grouping
```

```
        indices
        * e.g., if N(WVL)=6, and SYSGR=[0,1,2,1,0,2], then rows
          1 and 3 are modified in conjunction (i.e., if #1 is
          changed by a factor 10^(r*SYSDEV[1]), then #3 is changed
          by a factor 10^(r*SYSDEV[3]), where r is the same for both),
          2 and 5 are modified in conjunction, but 0 and 4 are
          _not_ tied together and are independently varied
        * if SYSDEV is not defined, SYSGRP is ignored
        * size must match that of FLX, but if smaller, the missing
          elements are set to -1, i.e., are all tied together
          - so if SYSGRP is not defined at all, all the rows are
            tied together, and only SYSDEV[0] is used
demrng  [INPUT] array containing the allowed range of variation
        in DEM: DEMRNG(i,0) < DEM(i) < DEMRNG(i,1)
        * if not supplied, DEMRNG(*,0)=DEM/1e5 & DEMRNG(*,1)=DEM*1e5
        * if DEMRNG(k,0)=DEMRNG(k,1), this component is frozen
abrng   [INPUT] as DEMRNG, for ABUND.
        * if not supplied, ABRNG(*,0)=ABUND(*)=ABRNG(*,1)
xdem    [INPUT] bin boundaries for accumulating the realized DEMs
        * if scalar or 1-element vector, taken to be bin-width in log10
        * if not specified, runs from min(DEMRNG) to max(DEMRNG)
          in steps of 0.1 in log10
xab     [INPUT] bin boundaries for accumulating the realized ABUNDs
        * if scalar or 1-element vector, taken to be bin-width
        * if not specified, runs from min(ABRNG) to max(ABRNG) in
          steps of 0.1 in log10
xnsig   [INPUT; default=3] force a minimum search range of
                XNSIG*sigma_determined_internally
nsim    [INPUT; default: 100; min=10] # of batches
nbatch  [INPUT; default: 10; min=5] # simulations per batch
nburn   [INPUT; default=NSIM/10] max # of simulations to run
        before assuming that solution has stabilized
        * note that this is a maximum -- if burn-in is achieved
          earlier, the algorithm will move on to standard MCMC
nosrch  [INPUT] if set, does *not* carry out a (crude) grid search
        for a best guess initial starting point.
        * if set to a IDL savefile name, restores the best guess
          from said file
savfil  [INPUT] if set, saves all variables to an IDL save file
        prior to exit
        * if not a string, or is an array, saves to 'mcmc.sav'
bound   [INPUT; default=0.9] confidence interval of interest
        * minimum=0.1
hwhm    [INPUT] if set, returns the upper and lower Half-Widths
        at Half-Maxmimum in DEMERR and ABERR.
```

```
demerr  [OUTPUT] confidence bounds on MAP estimates of DEM
        * DEMERR(*,0) are lower bounds, DEMERR(*,1) are upper bounds
aberr   [OUTPUT] confidence bounds on MAP estimates of ABUND
        * ABERR(*,0) are lower bounds, ABERR(*,1) are upper bounds
smoot   [INPUT] if set, multiplies the local length scale determined
        using FINDSCALE by SMOOT.
smooscl [INPUT] if set, bypasses FINDSCALE and uses the scales input
        via this keyword as the local length scales
        * _must_ be in units of [bins], not delta_T, dlogT, or whatever
        * if vector and size matches LOGT, then uses the input values
          as the [bins] over which to smooth
        * otherwise assumes constant smoothing scale corresponding to
          first element
loopy   [INPUT] if set, "smooths" by converting to T^3/2, loop-like DEM
        * use keyword SLOOP to change the index
        * set to a number >1 to limit the maximum number of
          temperature components
spliny  [INPUT] if set, "smooths" by making a spline interpolation
sampenv [INPUT] envelope from which to sample the temperature bins.
        this is a way to avoid T bins which may have no data and thus
        work more efficiently to explore just that part of the parameter
        space that is interesting.
        * if not given, or is not understandable, assumed to be the
          simple envelope of all the emissivities in EMIS
        * if set to:
          'ONLYRAT' - then the envelope is computed taking
                into account the ratios
          'INCABUND' - then use emissivities weighted by the abundance
          'FLAT' - then make no distinction between different T bins
          'INCALLT' - then force a look at all T bins at a 1% level
          'STEPPED' - then zero everywhere EMIS does not cover the
                T range, and at least 25% of maximum where it does
          'COVER' - then zero everywhere EMIS does not cover the
                T range, and constant where it does
        * NOTE: multiple SAMPENV settings may be specified, and
          they will be applied in the above sequence -- whether the
          sequence makes sense or not is up to the user to determine
        * if set to a numerical array the same size as LOGT, then
          simply use that array as the envelope
        * NOTE: recommend using 'COVER' or 'STEPPED' for EMIS
          derived from broad-band filters such as for solar data
storpar [OUTPUT] parameters from the MCMC chain
storidx [OUTPUT] index of the parameters stored in STORPAR
stordem [OUTPUT] DEMs calculated from the parameters for given STORIDX
bestpar [OUTPUT] best values of the parameters as found in the chain
```

```
 simprb  [OUTPUT] the metric at the end of each batch, an array of size
         (NSIM+1), with last element containing the value for the best-fit
 simdem  [OUTPUT] the DEM at the end of each batch, an array of size
         (NT,NSIM+1), with last column containing the best-fit
 simabn  [OUTPUT] same as SIMDEM, but for abundances
 simflx  [OUTPUT] same as SIMDEM, but for the fluxes with each [DEM,ABUND]
 simprd  [OUTPUT] same as SIMFLX, but for the ratios, only if ONLYRAT is used
mixsstr [I/O]  Set to turn on line deblending via MIXIE().
         Line (and/or conitnuum) structure of
         RD_LIST() format containing info (wvl,emis,z) on
         lines (and/or continuum) for which possible contaminants will be identif
         and for which correction factors  will be
         calculated. Assumes that ion equilibria are
         included. (see RD_LIST() keyword INCIEQ).
mixlstr [I/O] Line (and/or conitnuum) structure of
         RD_LIST() format containing info (wvl,emis,z) on
         lines (and/or continuum) for which possible contaminants will be identif
         and for which correction factors  will be
         calculated. Assumes that ion equilibria are
         included. (see RD_LIST() keyword INCIEQ)
         set to turn on line deblending via MIXIE().
 verbose [INPUT] explicitly set to 0 to avoid making plots
         * THINK TWICE before doing this!!
 _extra  [INPUT] pass defined keywords to subroutines
        ties    [ADJUSTIE] tie abundance parameters
         temp    [LINEFLX] assume input LOGT are actually T [K]
         noph    [LINEFLX] compute fluxes in [ergs/s/...] not [ph/s/...]
         effar   [LINEFLX] effective area in [cm^2]
         wvlar   [LINEFLX] wavelengths at which EFFAR are defined [Ang]
         nhne    [LINEFLX] ratio of N(H)/n_e, assumed to be 0.83
         softlim [LIKELI] allow for upper limits to be softly defined
         steps   [VARSMOOTH] assume stepped averaging from smoothing scale
         weight  [VARSMOOTH] allow weighting of adjacent points in boxcar smooth
         type    [VARSMOOTH] functional form to smooth with, see X3MODEL/LIBMODEL
         boxcar  [VARSMOOTH] controls smoothing behavior at endpoints
         betap   [MK_LORENTZ/MK_SLANT] index of beta-profile
         angle   [MK_SLANT] angle of tilt
         fwhm    [MK_GAUSS/MK_LORENTZ/MK_ROGAUSS] return FWHM rather than
                 sigma or core width
         sloop   [LOOPEM] slope of loop DEM
         dfx_mul [GENERATIO] multiplicative delta_Fx to compute partials
         dfx_add [GENERATIO] additive delta_Fx to compute partials
         choice  [FINDSCALE] flag to choose algo to find the scales
         pick    [FINDSCALE] flag on hoe to collapse scales from 2D to 1D
         crunch  [FINDSCALE] collapse 2D to 1D prior to finding scales
```

```
                half    [FINDSCALE] return half-scale
                eps     [FINDSCALE] a small number
               mproxy  [MIXIE]  see MIXIE()
               mplet   [MIXIE]  see MIXIE()

restrictions
        requires subroutines
        -- KILROY
        -- GETABUND
        -- MCMC_DEM_ONLY [LINEFLX, LIKELI, MK_DEM, VARSMOOTH, LOOPEM]
        -- LINEFLX [GETABUND, WHEE]
        -- LIKELI [SOFTLIM]
        -- FINDSCALE [WVLT_SCALE [ROOFN]]
        -- MK_DEM
        -- LOOPEM
        -- VARSMOOTH
        -- GENERATIO
       -- ADJUSTIE

history
        vinay kashyap (Apr 97)
        added keyword SMOOT (VK; May97)
        added keyword LOOPY and replaced call to VARSMOOTH by call to
          MK_DEM (VK; Nov01)
        added keyword ONLYRAT and call to GENERATIO (VK; Nov'02)
        numerous bug fixes (VK/LL; Dec'02)
        ratio stuff was not being passed on to MCMC_DEM_ONLY; now stores
          simulated DEMs, ABUNDs, fluxes, and ratios (VK/LL; Jan'03)
        added keywords SAMPENV,STORPAR,STORIDX,SIMPRB,SIMDEM,SIMABN,
          SIMFLX,SIMPRD,NBURN (VK; Feb'03)
        added keyword SMOOSCL (VK; May'03)
        added keyword SPLINY (VK; Jun'03)
        added _extra to call to PRED_FLX (VK; Jul'03)
       bugfix sampenv ONLYRAT keyword handler included (VK/LL:Dec'03)
       bugfix sampenv was ignored if input as array (LL: Dec'03)
       bugfix nemt is undefined in logt keyword handler(LL:Dec'03)
       added BRNSTR in call to MCMC_DEM_ONLY() to enable deblending
         via MIXIE() (LL:Jan'04)
       bugfix abndupdt, a MIXIE keyword  now set by default and
         turned off if abrng keyword is set (LL:Mar'04)
        added keyword XNSIG (VK; Jun'04)
        bug fix: compute DEMs for /LOOPY and /SPLINY on correct T grid
          (VK; Jan05)
        allow limiting number of T components for LOOPY (VK; Mar05)
       added call to adjustie and MIXLSTR keyword (LL; Jun05)
```

```
added LCOMP to call to MK_DEM, check to make sure FSIGMA
  are all +ve definite, and a warning if ABUND[0] is not 1,
  force single T deviate per iteration for LOOPY (VK; Jun05)
bug correction: if no update found in a batch, was still
  storing DEM from that batch; now corrected (VK; Jul05)
added options COVER and STEPPED for SAMPENV (VK; Jan06)
include extra kick at NPAR+1 to unfreeze parameters in case
  of LOOPY getting stuck; also bug correction for NPAR+1
  selections in regular case if some T bins are frozen
  (VK; Apr06)
added keyword VERBOSE (VK; Jun07)
added keyword BESTPAR (VK; Nov08)
added keywords SYSDEV and SYSGRP (VK; Apr09)
changed how stddev of proposal distr. is calculated, from using
  histograms to using unbinned distr.; added keyword STORDEM and
  now STORPAR outputs unsmoothed parameters (VK; Apr12)
```

B.130 MCMC_DEM_ONLY [/fitting]

```
procedure       mcmc_dem_only
        dedicated subroutine to MCMC_DEM, written only to make MCMC_DEM
        more readable.  this part decodes the parameters, calls LINEFLX
        to get predicted fluxes and LIKELI to get the likelihood.

syntax
        mcmc_dem_only,pars,fobs,fsigma,ulim,line,logT,wvl,Z,scale,ff,prb,$
        dem,abnd,loopy=loopy,/spliny,nrats=nrats,onlyrat=onlyrat,obsdat=obsdat,$
        obssig=obssig,idxflx=idxflx,prddat=prddat, _extra=e

parameters
        pars    [I/O] parameters {DEM(T),ABUND}, ABUND can be updated
                      via MIXIE()
        fobs    [INPUT] observed fluxes
        fsigma  [INPUT] errors on FOBS
        ulim    [INPUT] integer array specifying upper limits (1: UL, 0: not)
        line    [INPUT] emissivities of lines identified with FOBS
        logT    [INPUT] temperatures at which LINE is defined
        wvl     [INPUT] wavelengths at which fluxes are observed
        Z       [INPUT] atomic numbers of each id'd line
        scale   [INPUT] smoothing scale
        ff      [OUTPUT] predicted fluxes
        prb     [OUTPUT] likeihood (could be chi^2)
        dem     [OUTPUT] DEM
        abnd    [OUTPUT] abundances

keywords
        loopy   [INPUT] if set, "smoothing" by converting to T^3/2 DEM
        spliny  [INPUT] if set, "smooth" by using spline interpolation
        onlyrat [INPUT]
        obsdat  [INPUT]
        obssig  [INPUT]
        idxflx  [INPUT]
        nrats   [INPUT]
        verbose [INPUT]
        _extra  [INPUT] pass defined keywords to LINEFLX (TEMP, EFFAR,
                WVLAR, KEV, REGRID), LIKELI (SOFTLIM, CHI2, RCHI, BINOM),
                VARSMOOTH (STEPS, WEIGHT, TYPE), LOOPEM (SLOOP), or
               ADJUSTIE (TIES)

restrictions
        no error checking is done because it is assumed that this is
        called from and only from MCMC_DEM
```

```
history
        vinay kashyap (May97)
        changed SIGMA to FSIGMA; added DEM and ABND to parameter list
          (VK; AugMM)
        added keyword LOOPY, and switched out VARSMOOTH for MK_DEM
          (VK; Nov01)
        added keywords ONLYRAT,OBSDAT,OBSSIG,IDXFLX,NRATS and call to
          GENERATIO (VK; Dec'02)
        numerous bug fixes (liwei lin/VK; Dec'02)
        added keyword PRDDAT (VK; Jan'03)
        added keyword SPLINY (VK; Jun'03)
       added keywords BRNSTR,MIXLSTR,MIXSSTR,BRNSTR,XFRAC, as well as
         abundance updating via parameter  PARS. PARS now
         contains updated abundances from MIXIE() on
         output if MIXIE() keyword ABNDUPDT is set (LL; Jan'04)
        bug fix: compute DEMs for /LOOPY and /SPLINY on correct T grid (VK; Jan05)
        allow limiting number of T components for LOOPY (VK; Mar05)
       added call to ADJUSTIE to tie ABUND values (LL; Jun05)
        added keyword VERBOSE (VK; Jun07)
```

B.131 MCMC_DEM_POST [/fitting]

```
function        mcmc_dem_post
        returns the best-fit values and the error ranges for the
        various parameters in a structure of the form:
        {LOGT, NUEFF, MINCHISQ, BOUND,$
        DEM_BEST,DEM_MODE,DEM_HPD,DEM_EQT,DEM_MEDIAN,DEM_RNG,DEM_ENV,DEM_FRAC,$
        AB_BEST,AB_MODE,AB_HPD,AB_EQT,AB_MEDIAN,AB_RNG,AB_ENV}

syntax
        mcmcstr=mcmc_dem_post(savfil,clev=clev,verbose=verbose)

parameters
        savfil  [INPUT; required] full pathnmae to the save file
                that is written out by MCMC_DEM()

keywords
        clev    [INPUT] set this to override the confidence level at
                which to determine the bounds on the parameters
                * default is to use EBOUND in SAVFIL
                * if < 0, abs value is used
                * if > 1 and < 100, then assumed to be given as a percentage
                * if > 100, then 1-1/CLEV is used as the true value
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

output description
        LOGT:   the temperature grid
        NUEFF:  the effective degrees of freedom for adopted smoothing
        MINCHISQ:       the minimum chisq achieved
        BOUND:  the confidence level at which to compute errors
        *_BEST: best-fit parameter value
        *_MODE: mode of parameter
        *_MEDIAN:       median of parameter
        *_HPD: highest probability-density range at confidence BOUND
        *_EQT: equal-tail range at confidence BOUND
        *_RNG: half-tail range at confidence BOUND measured from best-fit
        *_ENV:  envelope range of best BOUND fraction (by chisq)
        DEM_FRAC:       number of hits at each temperature bin
        HELP:   a description of the output

subroutines
        MODALPOINT
        HIPD_INTERVAL
        EQT_INTERVAL
```

```
history
        vinay kashyap (Apr2006)
        bug fix: simprb was being compared to chisqcvf rather than half that
          (LL; May2006)
```

B.132 MCMC_DEM_WHISKERPLOT [/fitting]

```
procedure       mcmc_dem_whiskerplot
        make a nice whisker plot of the DEM solutions from MCMC_DEM.
        The central box covers 50% of the solutions at each temperature,
        and the whiskers cover the full range.  Individual points are
        also overplotted, along with the mode at each temperature, and
        the full DEM(T) that corresponds to the mode at each T, as well
        as the overall best-fit solution.

syntax
        mcmc_dem_whiskerplot,savedem,epsroot=epsroot,thrfrac=thrfrac,thrnum=thrnum,/dvdt,$
        logT=logT,simdem=simdem,storidx=storidx,$
        cbox=cbox,cwhisk=cwhisk,cdots=cdots,cbest=cbest,cmode=cmode,cgrey=cgrey,cnum=cnum,$
        labul=labul,labur=labur,labll=labll,lablr=lablr,yul=yul,yur=yur,yll=yll,ylr=ylr,$
        xrange=xrange,yrange=yrange,xtitle=xtitle,ytitle=ytitle,title=title,cthick=cthick,csize=cs
        verbose=verbose,$
        other PLOT keywords

parameters
        savedem [INPUT; required] this is the name of the file where
                MCMC_DEM saves its state
                * if set to 'NONE' or 'none', looks for keywords
                  LOGT,SIMDEM,STORIDX for input

keywords
        epsroot [INPUT] root name for encapsulated postscript file
                * if set, puts the plot in EPSROOT.eps
        thrfrac [INPUT] the fraction of the maximum of the histogram
                of selected temperatures in MCMC iteration to use
                * this looks at how many times a given temperature index
                  is present in the MCMC trace
                * default is 0.05
                * if 0, assumes default
                * if >1, the reciprocal is used
                * if <0, abs value fraction is taken of the _total_ number of
                  of iterations, not of the max amongst different T
                * NOTE: the greater of THRFRAC and THRNUM is used
        thrnum  [INPUT] the minimum _number_ of points of a given T
                that should be present
                * default is 100
                * if 0, assumes default
                * if <0, abs value is used
```

```
        * there is a hardcoded minimum of 5
        * NOTE: the greater of THRFRAC and THRNUM is used
dvdt    [INPUT] if set, converts the units of DEM to [.../degK]
        before plotting
        * MCMC_DEM computes DEMs as dV/dlogT == dV/dlnT/ln(10) == (T/ln(10))*dV/
          so if this is set, divides SIMDEM by T/ln(10)

logT    [INPUT] logT grid, used iff SAVEDEM=’NONE’
simdem  [INPUT] simulated DEMs, used iff SAVEDEM=’NONE’
storidx [INPUT] logT index selections during MCMC iterations, used iff SAVEDEM=’

cbox    [INPUT; default=none] color index for box
        * a color is applied only if set
cwhisk  [INPUT; default=3] color index for whiskers
cdots   [INPUT; default=4] color index for dots on whiskers
        * recommend setting this to 0 for postscript output
cbest   [INPUT; default=8] color index for best-fit curve
cmode   [INPUT; default=1] color index for mode marker
cgrey   [INPUT; default=95] color index for example curves
        * set CWHISK,CDOTS,CBEST,CMODE,CGREY explicitly to 0 to prevent plotting
cnum    [INPUT; default=9] color index for marking the number of samples

labul   [INPUT] LABel to put on Upper Left of plot
labur   [INPUT] LABel to put on Upper Right of plot
labll   [INPUT] LABel to put on Lower Left of plot
lablr   [INPUT] LABel to put on Lower Right of plot
yposul  [INPUT] y-location of LABUL
yposur  [INPUT] y-location of LABUR
yposll  [INPUT] y-location of LABLL
yposlr  [INPUT] y-location of LABLR
        * there are no equivalent XPOS because they can always be adjusted with
          judicious use of spaces within the label strings

xrange  [INPUT] passed on to PLOT without comment
yrange  [INPUT] passed on to PLOT without comment
xtitle  [INPUT] passed on to PLOT without comment
ytitle  [INPUT] passed on to PLOT without comment
title   [INPUT] passed on to PLOT without comment
cthick  [INPUT] line thickness
        * default is 2, unless EPSROOT is set, when it is 5
csize   [INPUT] character and symbol sizes
        * default is 2, unless EPSROOT is set, when it is 1.2

verbose [INPUT] controls chatter
_extra  [INPUT ONLY] pass defined keywords to subroutines
```

```
requires subroutines
        PEASECOLR
        MODALPOINT

history
        Vinay Kashyap (2015sep)
        bug fix with lower bound of whisker box not including mode; ensure clean quit
          when NSIM is small (VK; 2015nov)
```

B.133 MCMC_PLOT [/fitting]

```
procedure   mcmc_plot

       This procedure plots Monte Carlo Markov-Chain DEM
       reconstruction results. One sigma uncertanties are
       shown as a grey scale (or otherwise specified) with a gradient
       determined by how the N simulated DEMs (which  ordain the one sigma
       confidence intervals) are distributed about the best-fit DEM.
       Alternatively, the gradient may be determined by the p(D|M) of
       each DEM.

       NOTE: ALL REQUIRED INPUTS ARE STANDARD MCMC_DEM() OUTPUTS

parameters
       logt    [INPUT; required] array which determines the mid-bin values
               of the temperature grid on which DEM is defined
       simdem  [INPUT; required] two dimensional array [NT, NSIM+1]
               simulated arrays, the last of which is the best fit.
       demerr  [INPUT; required] confidence bounds on MAP estimates of DEM
                * DEMERR(*,0) are lower bounds, DEMERR(*,1) are upper bounds
       simprb  [INPUT; required] the p(D|M) of each DEM
       schme   [INPUT; optional] string designating plotting strategy. Can be:

                    'ERRBAR'      - Just plot error bars using DEMERR (default)
                    'DIFFERENCE'  - Colors are determined by
                                    difference of each DEM simulation
                                    bin value from the best fit DEM
                                    values. Colors are polyfilled
                                    between simulated values within
                                    each  logT bin.
                    'INDEX'       - Colors are determined by the order in which
                                    they appear (starting from the best
                                    DEM outwards). Colors are polyfilled
                                    between simulated values within each
                                    logT bin.
                    'SPLINE INDEX'- Colors are determined by the order in which
                                    they appear (starting from the best
                                    DEM outwards). Colors are polyfilled
                                    between simulated values within each
                                    logT bin. Spline interpolated
                                    curves are plotted in lieu of
                                    histogram style bins
                    'PROB'        - Colors are determined by the p(D|M)
                                    of each DEM. Values are oploted
```

```
                      with thick bars between simulated
                      values within each  logT bin.
       'SPLINE'     - Simulated DEMS are spline
                      interpolated and colors determined
                      by the p(D|M) associated with
                      each. Each spline is oploted onto
                      a best-fit dem plot.
       'NICE'       - For each temperature bin, a median simulation
                      value is calculated. The median values will be
                      plotted (dotted line) together with the DEM
                      simulation (solid line) that best fits them.
                      Uncertainty will be displayed by plotting the MIN and
                      MAX simulation values for each
                      bin. (dashed lines) Use the SLECT and CLEV
                      keywords to limit which simulations to include.
       'NICE DIFFERENCE' - A combination of DIFFERENCE and NICE
                      Colors are determined by
                      difference of each DEM simulation
                      bin value from the best fit DEM
                      simulation to the median value as
                      described in the 'NICE'
                      definition.  Colors are polyfilled
                      between simulated values within
                      each  logT bin.
       'NICE INDEX'    - A combination of DIFFERENCE and NICE
                      Colors are determined by the order in
                      which they appear (starting from the
                      best fit simulation to the median value
                      as described in the 'NICE' definition.
                      Colors are plyfilled between simulated
                      values within each logT value.
       'SUPPORT'       - Just plot upper and lower bounds of the
                      support in each bin given the set of
                      selected DEMs (see SLECT keyword)

   storidx [INPUT ; optional]  from MCMC_DEM(), contains the indices of
                      of all the sampled parameters stored in
                      STORPAR. If a temperature bin has not been
                      adequately sampled ( default threshold is
                      nsim*0.10 ), then the bin is not
                      plotted. Only bins at the ends of the Log
                      T range are thrown out. (see SAMPCT keyword)

keywords
   slect    [INPUT]  choose which DEMS to include default=[2]
```

```
                  1:ALL     -Include all simulations
                  2:CHI^2   -Limit simulations to best 50% p(D|M) DEMS
                              (default = 50, use CLEV keyword to toggle)
                  3:Errors  -Limit simulations to values in each bin that
                             lie  within the specified confidence bounds (demerr)
clev     [INPUT]  To be used in conjunction with slect=2,
                  determines what percentage by which to limit
                  sims in p(D|M)
col_rng  [INPUT]  index range in the color table to cover.
                  gradient will taper
                  off from 0 (darkest) to
                  col_rng (brightest). (maximum is 255)
col_tabl [INPUT]  color table on which to base the gradient
col_shft [INPUT]  number of color indices by which to shift
                  brighter(or darker if black background), col_rng will
                  be adjusted accordingly
ps_fil   [I/O]    name of ps file to output. If not specified
                  plot will be sent to current device
demslope [I/O]    if set, linear fits are made to the simulated DEMs
                  in a temperature range specefied by keyword SLOPEDT
                  and DEMSLOPE will return the mean slope. The
                  distribution of slopes will be plotted
slopesig [OUTPUT] if set, returns standard deviation slopes of
                     linear fits
slopedt  [INPUT]  temperature range specified in Log(T[K]) within which
                  to make linear fits to DEM
                      * if 2-element array, then SLOPEDT =
                        [min(SLOPEDT),max(SLOPEDT)]
                      * if 1-element array, then SLOPEDT =
                        [SLOPEDT, logt(max(bestdem)))] or
                        [SLOPEDT, logt(min(bestdem))]
                        if logt(max(bestdem)) eq SLOPEDT
                      * if not set, then SLOPEDT = [min(logt),
                        logt(max(bestdem))] or [SLOPEDT,
                        logt(min(bestdem))] if logt(max(bestdem)) eq SLOPEDT
sampct   [INPUT]  when STORIDX is input, set this to threshold for T
                  bin to be included. Expressed as percent of nsim. (default = 10
goodt    [OUPUT]  if STORIDX is input, GOODT will containt the log T
                  range used to plot
tbuff    [INPUT]  if STORIDX is set, TBUFF can be set to the number of
                  bins to add on either side of the temperature range
                  determined with STORIDX
monochrom[INPUT] set this to a value between 0 and 255 to make
                 all the plots in this color.
```

```
restrictions
       The SPLINE and SPLINE INDEX plotting schemes may not be used with slect = 3
       NOT tested yet with positional parameters e.g. !P.POSITION

subroutines
       mid2bound
       peasecolr
       stample
history

    (LL 6/03)
    ADDED 'Nice' and 'Nice Difference' option for schme parameter (LL 2/04)
    ADDED call to STAMPLE (LL 2/04)
    ADDED call to PEASECOLR to restore settings after loadct (LL 2/04)
    ADDED recognition of weather plot background is black or white
          and and adjustment to col_shft accordingly (LL 2/04)
    FIXED SPLINE INDEX schme so that it actually polyfills and doesn't just
          oplot i.e. no more 'speckled' look. Also added a NOERASE
          plot statement to prevent polyfills from hiding tickmarks (LL 2/04)
    FIXED color schemes so that the background color recognition
          scheme would work with both psuedo color and true color (LL 2/04)
    REMOVED PEASECOLR because IDL dynamically updates the x windows
          display whith the current color table (LL 2/04)
    BUGFIX variables CUPBND and CLOBND should be of size nT not nkpt (LL 3/04)
          if this affected you the routine would have crashed.
    ADDED 'NICE INDEX' scheme. (LL 10/04)
    CHANGED 'PROB' color schme so best-fit is plotted brightest for
        'x' and darkest for 'ps', order in which bars plotted matters! (LL 9/05)
    BUGFIX now compatible with multiple plots per page via !p.multi (LL 10/05)
    ADDED 'SUPPORT' scheme and keywords MONOCHROM and OUTMED (LL 11/05)
    CHANGED col_tabl behavior, only loads col_tabl when set, so default is
           now whatever is currently loaded and not B/W Linear (LL 11/05)
    ADDED keywords DEMSLOPE, SLOPSIG, SLOPEDT and parameter STORIDX (LL 1/06)
    CHANGED behavior of output to be not in color if COL_TABL is 0 (VK 8/07)
```

B.134 MDLPOLY [/stat]

```
function        mdlpoly
        compute and return the minimum description length statistic obtained
        by fitting polynomials of different orders

syntax
        mdl=mdlpoly(ydat,xdat,yfit=yfit,kord=kord,pfit=pfit,$
        kmin=kmin,kmax=kmax,verbose=verbose)

parameters
        ydat    [INPUT; required] data to be fit
        xdat    [INPUT] abscissae for YDAT
                * if not given, assumeed to be lindgen(n_elements(YDAT))

keywords
        yfit    [OUTPUT] best-fit function corresponding to KORD and PFIT
        kord    [OUTPUT] polynomial order at which minimum MDL is achieved
        pfit    [OUTPUT] parameters for the best-fit function
        kmin    [INPUT] minimum polynomial order to consider
                * default is 0, corresponding to a constant
        kmax    [INPUT] maximum polynomial order to consider
                * default is the smaller of 10 or half the number of data points
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (2014sep)
```

B.135 MID2BOUND [/util]

```
function        mid2bound
        given mid-bin values, make an intelligent determination of
        the bin boundaries and return said boundaries.  given input
        array of N elements, returns array of N+1 elements.

        (actually, it's not all that intelligent.  uses spline interpolation)

syntax
        dm=mid2bound(m,/halfit,eps=eps,verbose=verbose)

parameters
        m       [INPUT; required] the mid-bin values from which to
                expand out

keywords
        halfit  [INPUT] if set, does the simpleminded thing, i.e.,
                puts the bin half-way between 2 points, and extrapolates
                by repeating previous bins
        eps     [INPUT] a small number, default is 1e-6
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (OctMM)
        added keyword HALFIT (VK; Jul01)
        default EPS had not been defined (VK; Jan05)
```

B.136 MIXIE [/]

```
function mixie

      MIXIE() computes correction factors for line blends given a set of
      intersting lines (LSTR) and a set of contaminants (SSTR).
      LSTR and SSTR should both be RD_LIST() style structures, which
      contain necessary line information (emissivity, wvl, z, etc.)

      MIXIE()  will identify which lines in SSTR are possible contaminants
      for each line in LSTR.  SSTR lines must meet two criteria to be
      included in the correction fraction computation of each LSTR line:

      1) The SSTR line lies within 2*NSIG*LWDT on either side of the LSTR line
      2) The total contribution of the line is  greather than a
         user-defined threshold: LTHR * INTENSITY OF LINE (LTHR is
         keyword input while intensity of the interesting line is
         computed internally.)

      Correction factors are estimated by computing complimentary
      error functions with an IDL native routine ERRORF() for each
      contaminant (i.e. gaussian  profiles are assumed). If an
      OGIP-compliant rmf is fed via the RMFSTR keyword, the rmf
      will be used to estimate correction factors rather than
      ERRORF() <<RMFSTR NOT IMPLEMENTED YET>>.
      The correction factor is defined as follows:

        CORRECTION FACTOR = INT / (INT + total contributions from blends)
        where INT = intensity of interesting line

      If set, the MIXSTR keyword can return a structure containing the
      identification, emissivities, and wavelengths of each contaminant
      line, together with LSTR and SSTR indices which match contaminated line
      to contaminant (VICTIM and CULPRIT respectively). Also to be contained in
      the structure are the fractional contributions CONTAMFRAC and intensity of
      of the contaminant CONTAMINT (i.e. the total contribution of the contaminant is
      CONTAMINT*CONTAMFRAC). The fields in the output structure will be the
      the following (see KEYWORDS for individual descriptions):

      {CONTAMFRAC,CONTAMINT,VICTIM,CULPRIT,WVL,IDTAG}


syntax
      x=mixie(lstr,sstr,lwdt=lwdt,nsig=nsig,lthr=lthr,rmfstr=rmfstr,mixstr=mixstr,$
           contamfrac=contamfrac,victim=victim,culprit=culprit,contamint=contamint,$
```

```
          idtag=idtag,vint=vint,mproxy=mproxy,mplet=mplet,_extra=e)

parameters
      lstr   [INPUT;required] Line (and/or conitnuum) structure of
                      RD_LIST() format containing info (wvl,emis,z) on
                      lines (and/or continuum) for which possible contaminants will be identified
                      and for which correction factors  will be
                      calculated. Assumes that ion equilibria are
                      included. (see RD_LIST() keyword INCIEQ)
      sstr   [I/O]    Line structure of RD_LIST() format
                      containing user specified line contaminant suspect
                      info (wvl, emis, z).  Assumes that ion equilibria are
                      included. (see RD_LIST() keyword INCIEQ). If SSTR is
                      ommitted or undefined on input, the emissivity database will
                      read with RD_LIST() in the neccessary wavelength range
                      and including all species to create a list of  suspects.
                      If undefined on input, will contain RD_LIST() structure
                      on output.

keywords
      lwdt   [INPUT]  Std. deviation to be used as an estimate for
                      line broadening. Only contamination candidate profiles (sstr)
                      within 2*lwdt*nsig of the line in question will
                      be considered. May be single value or an array
                      with values for each element in LSTR. (must
                      match number of lines+continuum specified in LSTR)
                      default = 0.2 pixels
      nsig   [INPUT]  number of standard deviations from line of interest that
                      defines wavelength range within which to consider contamination.
                      If not set, the default is determined by LTHR.
                      May be single value or an array of values for
                      each element in lstr. If array, must match the
                      number of lines+continuum specified in LSTR.

                      So the range within which to consider
                      contamination for each line is defined as:

                         WRANGE = [LSTR.WVL(x)-LWDT*NSIG,LSTR.WVL(x)+LWDT*NSIG]

                      or if LWDT and NSIG are arrays:

                         WRANGE = [LSTR.WVL(x)-LWDT(x)*NSIG(x),LSTR.WVL(x)+LWDT(x)*NSIG(x)]

                      NOTE: To handle continuum measurements,
                      choose an NSIG which adequatly covers your
```

```
                continuum measurement wavelength range.


lthr   [INPUT]  fraction of intensity of the line of interest to
                use as cut-off intensity value for contaminant
                candidate fractional flux  contributions. default = 1e-4
                i.e. cutoff = (intnsity of line of interest)/100
rmfstr [INPUT]  <<NOT IMPLEMENTED YET>> structure containing OGIP compliant RMF. s
                RD_OGIP_RMF(). If set, contamination will be
                estimated by analyzing RMF instead of using ERRORF()
mixstr [OUTPUT] Structure containing detailed information for
                each pair of blends. The structure fields are
                the following:

                {CONTAMFRAC,CONTAMINT,VINT,VICTIM,CULPRIT,WVL,IDTAG}

                Each field can also be returned individually:

contamfrac [OUTPUT] array containting contribution fractions for
                each culpable line
contamint [OUTPUT] the contaminant line intensities
vint [OUTPUT] the victim line intensities
victim [OUTPUT] lstr index of intersting line being contaminated
culprit[OUTPUT] sstr index of indices of contaminant lines
idtag  [OUTPUT] string containing derscription of each culprit.
                the description is as follows:

                SPECIES, WVL, DATABASE, DESCRIPTION

                SPECIES gives atomic symbol and ionic
                state. WVL gives Wavelength. DATABASE gives
                the name of the DATABASE used. And DESCRIPTION
                gives the level designations and electron
                configurations of the transition. e.g.

                MgXII 8.41920 $CHIANTI (1s) 2S_1/2 (2p) 2P_3/2
                OVIII 18.9726 $CHIANTI (1s) 2S_1/2 (2p) 2P_1/2

vidtag [OUPUT]  string containing description of each victim
                and of same format as idtag
mproxy [INPUT] If a correction factor is needed for  a
                 multiplet measured using a single profile,
                 then MPROXY will specify the
                 multiplet components on input, and contain
                 the multiplet correction factor on output.
```

One component of the multiplet is to stand proxy for the group, and must be specified in LSTR.

MPROXY is defined as follows:

```
mproxy = strarr(2,N) or
mproxy = intarr(2,N)
```

N is the number of component-proxy relationships. Each component-proxy relationship is expressed as a two element string array. The first element should contain the LSTR index of the proxy. The second element should contain either the LSTR index of the component if it is included in LSTR, or a string description of the component in IDTAG format. One may optionally leave out:

- the DESCRIPTION field (e.g. just use SPECIES, WVL,DATABASE).
- the DATABASE and WVL fields (e.g. just use SPECIES, DATABASE)

On output, one correction factor will be given for each multiplet. The index of this correction factor in the output correction factor array is determined by the position of the proxy in LSTR. (See examples below)

mplet [INPUT] If blending between one or more multiplet components is already accounted for (e.g. a multiplet is modelled with serveral profiles) then MPLET can specify which multiplet component to leave out of the correction factor calculation of another component.

MPLET is defined as follows:

```
mplet = strarr(2,N)
mplet = intarr(2,N)
```

N is the number of component-component relationships. Each component-component

```
                   relationship is expressed as a two element
                   string array. The first element should
                   contain  the LSTR index of the component
                   whose correction factor is needed.
                   The second element should contain either the
                   LSTR index of the component to leave out, or
                   a string description of the component to
                   leave out in IDTAG format.
                   One may optionally leave out:

                      o the DESCRIPTION field (e.g. just use SPECIES, WVL,DATABASE
                      o the DATABASE or WVL fields (e.g. just use
                        SPECIES, DATABASE)

                Correction factors will not be calculated for
                lines specified to be left out of another
                lines' correction factor calculation. See
                examples below.
brnstr [I/O]    if mixie is to be run multiple times
                (e.g. using MCMC_DEM() ) the brnstr keyword can
                be set so that MIXIE() creates a sturcture
                containing internally calculated and recycle-able information, to 
                to subsequent MIXIE() calls. The structure is
                of the form:

                {FRACS,NDX,INTNDX,NSS,NNSS,PRXBSE,MULTSTR}

                where:

                FRACS   = float array contribution fractions of all non-exempt lin
                NDX     = long array  sstr index of each of thes lines
                INTNDX  = float array ndx and fracs index intervals corresponding 
                NSS     = float array lstr index of each victim
                NNSS    = long array  number of actual victims i.e. n_elements(nss
                PRXBSE  = long array  lstr index of the proxy in each proxy relati
                MULTSTR = STRUCT      rd_line structure of proxee in each relation

                When fed as input, BRNSTR allows MIXIE to avoid
                explicit contribution factor calculations for
                each suspect and cumbersome MPROXY and
                MPLET sting parsing.
DEM    [INPUT]  differential emission measure  at each T [cm^-5/logK]
                * default is a constant=1e14!!
                * if defined, DLOGT must also be specified
                * if LSTR and SSTR emissivity temperature grids do not
```

```
                   match  DLOGT then they are rebinned and
                   resampled so that they do
      DLOGT  [INPUT   logarithmic temperature grid at which DEM is defined
                   * default is 4.0 to 8.0 in steps of 0.05
      ABUND  [I/O]  abudnanaces relative to H abund(0) = 1
                    * abund(Z-1) contains the abundance for element Z
                   * default: Grevesse & Sauval(1998)
      ABNDUPDT [INPUT] If set, then abundnaces will be updated on
                   output. Abundances are calculated after taking
                   into account the correction factors
      OBSFLX [INPUT]  Obseved fluxes used to calculate abundnaces,
                   must match the number of correction factors
                   calculated
      OFSIG  [INPUT]  errors for OBSFLX
      MULTIPLET [INPUT] temporary, same as MPLET
      MULTPLET [INPUT]  temporary, same as MPLET
      MULTPROXY [INPUT] temporary, same as MPROXY
      MULTIPROXY[INPUT] temporary, same as MPROXY
      verbose[INPUT]  set verbosity
      _extra [INPUT]  pass defined keywords to
                    LINEFLX (DEM, ABUND, EFFAR, WVLAR, TEMP, NOPH, IKEV, REGRID)
                   RD_LIST (DBDIR,LOGP,PRES,N_E,EQFILE)

subroutines
      LINEFLX
      RD_LIST
      INICON
      CAT_LN
      REBINX
      MCMC_ABUND
      MCMC_EROR
      IS_KEYWORD_SET

restrictoins
      Is is suggested that LSTR and SSTR be read from a common
      database to avoid accidentally identifying a line as 'self blending'
      MIXIE() will NOT compute correct correction factors for DR lines.

usage examples

      * !LDBDIR = '$CHIANTI'
        Ne9   = rd_list('Ne9|[13.44,13.45]',/incieq,sep='|')
        cand  = rd_list('ALL|[12.44,15.45]',/incieq,sep='|')
        factor = mixie(Ne9,Cand,lwdt=0.008,nsig=5,lthr=0.02,contamfrac=contamfrac,victim=victim,$
                   culprit=culprit,contamint=contamint,idtag=idtag, mixst=mixstr)
```

```
  for j = 0,n_elements(culprit)-1 do print,mixstr.idtag(j), mixstr.contamint(j)* m

* !LDBDIR = '$CHIANTI'
  Ne9    = rd_list('Ne9|[13.44,13.45]',/incieq,sep='|')
  factor = mixie(Ne9,lwdt=0.008,nsig=5,lthr=0.02,contamfrac=contamfrac,victim=vict
           culprit=culprit,contamint=contamint,idtag=idtag, mixst=mixstr)
  for j = 0,n_elements(culprit)-1 do print,mixstr.idtag(j), mixstr.contamint(j)* m

* !LDBDIR = '$CHIANTI'
  T_components  = [6.6,6.9,7.4]           ; log(T[K]) components in the EM
  EM_components = [3.05,3.05,3.55]*1d11   ; Emission Measure [cm^-3]
  !DEM=mk_dem('delta', logT = !LOGT, pardem=T_components, indem=EM_components)
  Ne9   = rd_list('Ne9|[13.44,13.45]',/incieq,sep='|')
  factor = mixie(Ne9,lwdt=0.008,nsig=5,lthr=0.02,contamfrac=contamfrac,victim=vict
                culprit=culprit,contamint=contamint,idtag=idtag, mixst=mixstr,DEM=
  for j = 0,n_elements(culprit)-1 do print,mixstr.idtag(j),mixstr.contamint(j)* mi

* !LDBDIR = '$CHIANTI'
  N7  = rd_list('N7|[24.77,24.79]',/incieq,sep='|'); this list specifies a triplet
  mproxy = [ [2,1], [2,0] ]  ; We use one triplet component as proxy
  factor = mixie(N7,lwdt=0.008,nsig=5,lthr=0.008,contamfrac=contamfrac,victim=vict
           culprit=culprit,contamint=contamint,idtag=idtag,mproxy=mproxy,$
           mixstr=mixstr)

* !LDBDIR = '$CHIANTI'
  N7 = rd_list('N7|[24.7846,24.7847]',/incieq,sep='|'); this list contains one N7
  ; use MPROXY to specify N7 triplet components the one component are to stand pro
  mproxy = [['0','NVII 24.7793 $CHIANTI (1s) 2S_1/2 (2p) 2P_3/2'], $
            ['0','NVII 24.7844 $CHIANTI (1s) 2S_1/2 (2s) 2S_1/2']]
  factor = mixie(N7,njk,lwdt=0.008,nsig=5,lthr=0.008,contamfrac=contamfrac,victim=
                 culprit=culprit,contamint=contamint,idtag=idtag,$
                 mixst=mixstr,mproxy=mproxy )

* !LDBDIR = '$CHIANTI'
  N7 = rd_list('NVII|24.7847|$CHIANTI|(1s) 2S_1/2 (2p) 2P_1/2',sep='|',/incieq)
  ; use MPROXY to specify multiplet components the N7 at 24.7847 is to stand proxy
  mproxy = [['0','NVII 24.7793 $CHIANTI (1s) 2S_1/2 (2p) 2P_3/2'], $
            ['0','NVII 24.7844 $CHIANTI (1s) 2S_1/2 (2s) 2S_1/2']]
  factor = mixie(N7,j,lwdt=0.008,nsig=5,lthr=0.008,contamfrac=contamfrac,victim=vi
                 culprit=culprit,contamint=contamint,idtag=idtag,$
                 mixst=mixstr,mproxy=mproxy)
```

history

```
INTENDED as an improvement on bland() (LL:Jul'03)
NEW documentation new default nsig
```

```
  output now correction factor and not mixstr (LL:Aug'03)
BUG FIX: 1/SQRT(2) factor needed for arguement of ERRORF so
  lwdt is actually standard deviation
now prints message when no blends are found
now robust to theoretical lines (designated by
negative wavelengths)
BUG FIX correction factor calculations for multiple
victim list was incorrect (LL/DG:Sep'03)
ADDED keyword COMPOSITE to handle doublets (LL:Sep'03)
REPLACED keyword COMPOSITE with MUTLPROXY which now handles
  multiplets. ADDED related keyword MULTIPLET. (LL:Oct'03)
ADDED correction for 'over correction' i.e. a CULPRIT now
  cannot 'overdistribute' its flux when contaminating
  several VICTIMS at once. (LL:Oct'03)
CHANGED LTHR and NSIG so that they can handle arrays
  so correction factors for continuum measurements
  are possible (LL:Oct'03)
CHANGED default RD_LIST() call so that makes
  intelligent efficient wrange decision (LL:Oct'03)
ADDED BRNSTR keyword so that doesn't bog down
  MCMC_DEM(). ALSO included explicit intensity
  calculations so to avoid being slowed by LINEFLX(). (LL:Oct'03)
ADD ABNDUPDT keyword to toggle ABUND updating on
  output (LL:Jan'04)
CHANGED location of ABUNDANCE calculation so that it
  abundances get updated after contamination
  correction. ADDED keyword OFSIG (LL:Jan'04)
CHANGED MULTIPLET KEYWORD to MPLET and MULTPROXY to
  MPROXY (LL/DG:Jan'04)
 changed default of LTHR to 1e-4 (VK;Jan'04)
ADDED new MPLET and MPROXY format (LL:Feb'04)
BUG FIX over correction mechanism (i.e. considering
  situations of one culprit and two victims) was not
  working properly, leading to correction factors >
  1. (LL/DG:Feb'04)
BUG FIX total(contamfrac) gt 1 changed to
  n_elements(culprit) gt 1  (LL:Feb'04)
BUG FIX keywords idtag and mixstr were failing
  because need to handle with keyword_set()  not
  arg_present() (LL:Mar'04)
VARIOUS BUG FIXES
  sstr enteed as dummy argument = crash
  mixstr and idtag fix (LL:Mar'05)
 replaced direct reference to !LOGT with calls to
   defsysv and setsysval (VK; Apr'05)
```

```
        BUG FIX if sstr is one component only it was
          ignored. (LL:Apr'05)
updated for IDL5.6 keyword_set([0]) behavior change for vectors
  (VK; 20Mar2006)
```

B.137 MK_3MODEL [/fitting]

```
function        mk_3model
        generate and return a compound model using the supplied
        parameters of components made up of 3-parameter functions

syntax
        model=mk_3model(x,p,w,h,pder,group=group,delp=delp,type=type,/asis,$
        /allcomp,/fwhm,/normflx,missing=missing,betap=betap,vrot=vrot)

parameters
        x       [INPUT; required] points where model is computed
        p       [INPUT; required] positions of components
        w       [INPUT; required] widths of components
        h       [INPUT; required] heights of components
        pder    [OUTPUT] contains 2D array (NX,3*NPAR) of partial derivatives
                of each component wrt parameters

keywords
        group   [INPUT] grouping index of gaussians
        delp    [INPUT] offsets from first position value in group
                NOTE: GROUP and DELP are ignored if ASIS is set
                NOTE: if GROUP and DELP are not given, ASIS is set
        type    [INPUT; default='G*aussian'] what type of curve to generate
                for the model.  other possibilities are
                        L*orentzian
                        B*eta-profile (set "Beta=<value>")
                        SL*ant (set "slant=<angle>,<beta>")
                        R*otation-convolved Gaussian (set "rot=<velocity>")
                        P*ower-law (set "pw=<break>" or "pe=<break>")
                        I*dentity
                        U*ser_defined (not yet implemented), etc.
                        SI*nusoid (set "sin=<phase>")
                * if not defined, uses "Gauss"
                * if insufficiently defined, uses last defined as default
                  for the rest
        asis    [INPUT] the default action is to verify that the p are
                consistent with GROUP and DELP and to correct as necessary.
                if ASIS is set, this is not done.
        allcomp [INPUT] if set, returns the sum of components and also each
                component in a separate column of a 2D array of size
                (N_ELEMENTS(P)+1,N_ELEMENTS(X))
        _extra  [INPUT] pass defined variables to subroutine:-
                MK_GAUSS: MISSING, FWHM, NORMFLX
                MK_LORENTZ: BETAP, MISSING, NORMFLX
```

```
                MK_SLANT: ANGLE, BETAP, MISSING, NORMFLX
                MK_ROGAUSS: VROT, MISSING, FWHM, NORMFLX
                MK_IDENT: GIGO
                MK_SINUSOID: PHASE, MISSING

usage summary
        * call as a function
        * returns composite model by adding up components
        * input component parameters as sequence of arrays of
          positions, widths, and heights
        * input grouping info by 2 additional arrays
        * specify type of model with keyword TYPE
        * partial derivatives are computed only if ALL 5 parameters are
          supplied in call
        * example TYPEs:
          ’Gauss’, ’gaussian’, ’gau’, ’g’
          ’Lorentz’, ’lorentz’, ’lorentzian’, ’lor’, ’l’
          ’beta=1.8’, ’BETA=2.4’, ’B=2’, ’b 1.8’, ’b:1.8’, ’b   1.8’
          ’slant=45,1.8’,’SLANT=89,2.4’, ’s=10’,’s=,2’, ’s:3,2’
          ’rotational velocity=1e-4’, ’rot=30’, ’r:30e5’, ’R 1e-4’
          ’pe=1’, ’pw=12.3985’, ’pw@12.4’
          ’sin’, ’sin=90’

subroutines
        MK_GAUSS
        MK_LORENTZ
        MK_SLANT
        MK_ROGAUSS
        MK_POWLAM
        MK_SINUSOID
        KILROY

history
        vinay kashyap (Oct96)
        added parameter to return partial derivatives (VK; Nov96)
        added _EXTRA, added call to MK_LORENTZ (VK; Oct98)
        allowed setting BETAP for MK_LORENTZ via TYPE definition (VK; Dec98)
        added MK_SLANT calls (VK; MarMM)
        added MK_ROGAUSS, MK_POWLAM calls (VK; JunMM)
        changed name from MK_MODEL to MK_3MODEL (VK; Aug01)
        converted array notation to IDL 5; added MK_IDENT (VK; Apr02)
        changed keyword NORM to NORMFLX (VK; Oct02)
        added call to MK_SINUSOID (VK; Sep04)
```

B.138 MK_ABSORB [/fitting]

function mk_absorb

returns a multiplicative transmission factor for ISM absorption

syntax

```
ismcorr=mk_absorb(x,NH,pder, fH2=fH2,He1=He1,HeII=HeII,/Fano,/ikev,$
/wam,/bam,/mam,/noHeH,abund=abund,verbose=verbose)
```

parameters

x	[INPUT array; required] where the function must be computed
NH	[INPUT; default: 1e18] H column
pder	[OUTPUT; optional] partial derivatives of model wrt parameter at each X; calculated only if 3 parameters are supplied in call.

keywords

_extra	[INPUT ONLY] pass defined keywords to subroutines ISMTAU: FH2, HE1, HEII, FANO, IKEV, WAM, BAM, MAM, NOHEH, ABUND

description

wrapper to ISMTAU

subroutines

ISMTAU

history

vinay kashyap (Jul08)

B.139 MK_BBANG [/fitting]

```
function        mk_bbang
        returns the Planck function B(lam) [ph/s/cm^2/Ang] as a
        function of lam [Ang] at a given temperature T.

        In general,
        B(nu)d(nu) = (R/d)^2*(2*pi/c^2) * d(nu)*nu^2/(exp(h*nu/kT)-1)
        for lam=c/nu,
        B(lam)d(lam) = (R/d)^2 * (2*pi*c) * d(lam) * (1/lam^4)/(exp(hc/lam/k/T)-1)
                == A * (lam^-4/(exp(hc/lam/k/T)-1)) * d(lam)

syntax
        b=mk_bbang(x,norm,temperature,pder,verbose=verbose)

parameters
        x       [INPUT array; required] where B(X) must be computed
        norm    [INPUT; required] normalization for B(X)
        temp    [INPUT; required] temperature in [K]
        pder    [OUTPUT; optional] partial derivatives of model wrt parameters
                at each X; calculated only if 4 parameters are supplied in call.

keywords
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing

usage summary
        * call as a function

subroutines
        NONE

history
        vinay kashyap (Aug08; based loosely on 1993 blackbody.pro)
```

B.140 MK_BBKEV [/fitting]

```
function        mk_bbkev
        returns the Planck function B(E) [ph/s/cm^2/keV] as a
        function of E [keV] at a given temperature T.

        In general,
        B(nu)d(nu) = (R/d)^2*(2*pi/c^2) * d(nu)*nu^2/(exp(h*nu/kT)-1)
        for E=h*nu,
        B(E)dE  = (R/d)^2 * (2*pi/c^2/h^3) * dE*E^2/(exp(E/kT)-1)
                == A * (E^2/(exp(E/kT)-1)) * dE

syntax
        b=mk_bbkev(x,norm,temp,pder,verbose=verbose)

parameters
        x       [INPUT array; required] where B(X) must be computed
        norm    [INPUT; required] normalization for B(X)
        temp    [INPUT; required] temperature in [K]
        pder    [OUTPUT; optional] partial derivatives of model wrt parameters
                at each X; calculated only if 4 parameters are supplied in call.

keywords
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing

usage summary
        * call as a function

subroutines
        NONE

history
        vinay kashyap (Aug08; based loosely on 1993 blackbody.pro)
```

B.141 MK_BKNPOWER [/fitting]

```
function        mk_bknpower
        returns a broken Power-law function,
          p(X)  = NORM*(X/Xo)^gamma1 {X.le.PBREAK}
                = NORM*(PBREAK/Xo)^(gamma1-gamma2)*(X/Xo)^gamma2 {X.ge.PBREAK}

syntax
        p=mk_bknpower(x,norm,gamma1,gamma2,pbreak,pder,Xo=Xo,/nobreak,verbose=verbose)

parameters
        X       [INPUT; required] where p(X) must be computed
        norm    [INPUT; required] normalization
        gamma1  [INPUT; required] power-law index 1 (valid for 0<X<PBREAK)
        gamma2  [INPUT; necessary] power-law index 2 (valid for X>PBREAK)
        pbreak  [INPUT; necessary] break point
                * GAMMA2 and PBREAK are ignored if NOBREAK is set
        pder    [OUTPUT; optional] partial derivatives of model wrt parameters
                at each X; calculated only if 6 parameters are supplied in call.
                * array of size [N(X),4], with columns containing the partial
                  derivatives wrt NORM, GAMMA1, GAMMA2, and PBREAK respectively
                * to have it return PDER as [N(X),2] for a non-broken power-law
                  distribution, set NOBREAK (must have GAMMA2 and PBREAK in the
                  call, but they will be ignored)

keywords
        Xo      [INPUT] X-value at which which normalization is defined
                * default is 1.0
        nobreak [INPUT] if set, ignores GAMMA2 and PBREAK, essentially assuming
                that PBREAK=$\infty$
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

example
        x=(findgen(100)+1)/10. & p=mk_bknpower(x,1.,-1,-3,1.5) & plot,x,p,/xl,/yl
        p2=mk_bknpower(x,1.,-1,-3,1.5,/nobreak) & oplot,x,p2,line=2,thick=2

history
        Vinay Kashyap (Apr2008)
```

B.142 MK_DEM [/]

```
function        mk_dem
        returns DEM(T) [cm^-5]

syntax
        DEM=mk_dem(type,logT=logT,indem=indem,pardem=pardem,$
        sloop=sloop,xcur=xcur,ycur=ycur,loopy=loopy,lcomp=lcomp,$
        weight=weight,/gauss,verbose=verbose)

description
        this provides a way to generate DEMs quickly by various means.
        TYPE, a string variable, decides what sort of method to use to
        generate the DEM.  TYPE can be
                'help'
                'constant'      PARDEM = value
                'chebyshev'     PARDEM = coefficients
                'cursor'        PARDEM = max range in DEM
                'interpolate'   PARDEM = original logT
                'loop'          PARDEM = original logT
                'rebin'         PARDEM = rebinning scales
                'spline'        PARDEM = logT location of points
                'varsmooth'     PARDEM = smoothing scales
                'delta'         PARDEM = logT locations of EM components
        if a particular method requires an input DEM (e.g., 'interpolate'),
        feed that in via the keyword parameter INDEM.
        any parameters that the method requires go in via PARDEM (e.g., for
        'interpolate', PARDEM would be the temperature grid at which INDEM
        is defined).

parameters
        type    [INPUT; required] specifies what type of DEM.
                * if 'help', prints out available options

keywords
        logT    [INPUT; default=findgen(81)*0.05+4] log10(T [K]) at which
                to return DEM(T)
        pardem  [INPUT] parameters to be used to figure out the DEM.
                strictly case dependent.
        indem   [INPUT] for those cases that require an input/initial DEM
        xcur    [I/O] cursor X locations for the "cursor" option
        ycur    [I/O] cursor Y locations for the "cursor" option
        loopy   [INPUT] set to the maximum number of loop components
                that should be used for LOOP type DEMs
                * the top LOOPY EM values are kept as is, and the rest are
```

```
                  discarded
                * LCOMP lists the components that _must_ be kept
        lcomp   [INPUT] scalar or vector which contains the indices of
                the parameters that must be used to compute loops
                  -- if LCOMP exceeds LOOPY, only the largest of the
                     specified components that are within quota are kept
                  -- if LCOMP is less than LOOPY, the remaining slots are
                     filled in by the largest of the remaining components
        _extra  [INPUT ONLY] allows setting defined keywords to subroutines
                -- VARSMOOTH (WEIGHT, GAUSS, STEPS)
                -- LOOPEM (SLOOP, VERBOSE)

subroutines
        CHEBYSHEV, VARSMOOTH, LOOPEM

history
        vlk (Jun98)
        added option to interpolate (VK; Dec98)
        added option for variable scale rebin (VK; 1999)
        added option to interactively specify DEM (VK; MMJul)
        rebin option now works (VK; AugMM)
        added keywords XCUR and YCUR (VK; JanMMI)
        converted array notation to IDL 5 (VK; Apr02)
        added 'delta' option (VK; Mar03)
        now correctly converts EM to DEM for 'delta' (VK; Jun03)
        added keyword LOOPY; allows limiting the number of T components
          for LOOP (VK; Mar05)
        added keyword LCOMP (VK; Jun05)
        button press status now stored in !MOUSE, not !ERR (VK; Apr09)
```

B.143 MK_EFOLD [/]

```
function        mk_efold
        returns an exponential, NORM*exp(-x*SCALE)+YOFF

syntax
        f=mk_efold(x,norm,scale,yoff,pder,verbose=verbose)

parameters
        x       [INPUT array; required] where F(X) must be computed
        norm    [INPUT; required] normalization for exponential
        scale   [INPUT; required] reciprocal of e-folding decay scale
        yoff    [INPUT; required] offset from Y=0
                * NORM,SCALE,YOFF must be scalars or single-element
                  vectors -- extra elements, if present, are silently
                  ignored
        pder    [OUTPUT; optional] partial derivatives of model wrt parameters
                at each X; calculated only if supplied in the calling sequence.

keywords
        verbose [INPUT; default=0] controls chatter
        _extra  [JUNK] here only to prevent crashing

example
        x=findgen(100) & plot,x,mk_efold(x,100.,0.05,20.)

subroutines
        NONE

history
        vinay kashyap (Aug08)
```

B.144 MK_GAUSS [/fitting]

```
function        mk_gauss
        returns a gaussian G(X)

syntax
        g=mk_gauss(x,mean,sig,peak,pder,/fwhm,/normflx,missing=missing,/widebin)

parameters
        x       [INPUT array; required] where G(X) must be computed
        mean    [INPUT; default: mid_point(X)] position of peak
        sig     [INPUT; default: 0.1*range(X)] std. deviation
        peak    [INPUT; default: 1] max(G)
        pder    [OUTPUT; optional] partial derivatives of model wrt parameters
                at each X; calculated only if 5 parameters are supplied in call.

keywords
        fwhm    [INPUT] if set, the std.deviation is SIG/2.355
        normflx [INPUT] if set, the *normalization* is set to PEAK
        missing [INPUT] 3 element array to populate missing values of
                MEAN, SIG, and PEAK
        widebin [INPUT] if set, assumes that the bins are wide compared
                to the width of the Gaussian and returns the integral
                over the bins rather than just the value at each X
                * this part requires a call to MID2BOUND()
        _extra  [JUNK] here only to prevent program from crashing

description
    if NORMFLX == 0 && FWHM == 0:
        G(X)=PEAK*exp((X-MEAN)^2/2/SIGMA^2)
    if NORMFLX != 0 && FWHM == 0:
        G(X)=(PEAK/SIGMA/sqrt(2*!PI))*exp((X-MEAN)^2/2/SIGMA^2)
    if NORMFLX == 0 && FWHM != 0:
        G(X)=PEAK*exp(2.355^2*(X-MEAN)^2/2/SIGMA^2)
    if NORMFLX != 0 && FWHM != 0:
        G(X)=(2.355*PEAK/SIGMA/sqrt(2*!PI))*exp(2.355^2*(X-MEAN)^2/2/SIGMA^2)

usage summary
        * call as a function
        * generates gaussian model only at specified points X
        * needs MEAN, SIG, PEAK for complete specification

subroutines
        MID2BOUND
```

```
history
        vinay kashyap (Oct96)
        added PDER (VK; Nov96)
        added _EXTRA, changed SIGMA to SIG (VK; Oct98)
        big correction, if MEAN,SIG,PEAK are undefined on input (VK; JunMM)
        now works even if X are integers (VK; Jul01)
        converted array notation to IDL 5 (VK; Apr02)
        check to see if X is defined (VK; Sep02)
        changed keyword NORM to NORMFLX (VK; Oct02)
        added keyword WIDEBIN and call to MID2BOUND (VK; Apr11)
```

B.145 MK_IDENT [/fitting]

```
function        mk_ident
        returns the input as the output (hence the name)
        it is useful as an "identity translator" to make
        a model out of whatever input array.

syntax
        gy=mk_ident(x,y0,x0,m0,pder,gigo=gigo,verbose=verbose)

parameters
        x       [INPUT array; required] the points at which the output is
                computed.
        y0      [INPUT] the normalization of the output function
                * if not given, assumed to be 1.0
        x0      [INPUT] the zero-point of the output, i.e., how much to shift
                the input
                * if not given, assumed to be 0.0
        m0      [INPUT] a stretching factor
                * if not given, assumed to be 1.0
                * if Y0, X0, or M0 are arrays, only the first element is used
                * NOTE: for X0.NE.0 and M0.NE.1, the new functions are simply
                  interpolated from GIGO
        pder    [OUTPUT; optional] partial derivatives of model wrt parameters
                at each X; calculated only if 5 parameters are supplied in call.

keywords
        gigo    [INPUT] the array which goes into the output
                * if not given, assumed to be 0.*X+1.
                * if size does not match X, truncated or filled out with
                  1st element
        verbose [INPUT; default=0] controls chatter
        _extra  [JUNK] here only to prevent crashing

usage summary
        * call as a function

subroutines
        NONE

history
        vinay kashyap (Apr02)
```

B.146 MK_LCECLIPSE [/fitting]

```
function        mk_lceclipse
        returns the light curve of a hard eclipse of an almost point source

syntax
        lc=mk_lceclipse(x,midpt,hwidth,depth,pder,angfall=angfall,angrise=angrise,$
        ybase=ybase,yoff=yoff,/steep)

parameters
        x       [INPUT array; required] where LC(X) must be computed
        midpt   [INPUT; default: mid_point(X)] position of central point of eclipse
        hwidth  [INPUT; default: 0.1*range(X)] half-width of eclipse
        depth   [INPUT; default: 1] depth of eclipse
        pder    [OUTPUT; optional] partial derivatives of model wrt parameters
                at each X; calculated only if 5 parameters are supplied in call.
                * array of size [N(X),7], with columns containing the partial
                  derivatives wrt MIDPT,WIDTH,DEPTH,ANGFALL,ANGRISE,YBASE,YOFF

keywords
        angfall [INPUT; default=3*!pi/2] angle of lightcurve descent in radians
        angrise [INPUT; default=!pi/2] angle of lightcurve ascent in radians
                * NOTE: if STEEP is set, ANGFALL and ANGRISE are assumed to be slopes
                  and the default values are set to -1000 and +1000 respectively
        steep   [INPUT] if set, assumes that ANGFALL and ANGRISE are given as slopes
        ybase   [INPUT; default=0] baseline emission
        yoff    [INPUT; default=0] offset on the other side of eclipse
        eps     [INPUT; default=1e-6] a small number
        _extra  [JUNK] here only to prevent crashing

description
        The eclipse curve is:
        slp1=tan(ANGFALL) & slp2=tan(ANGRISE)
        LC(X)   = YBASE+DEPTH   for X < MIDPT-HWIDTH+DEPTH/slp1
                = slp1*X+(YBASE-slp1*MIDPT+slp1*HWIDTH)
                        for MIDPT-HWIDTH-DEPTH/slp1 .LE. X .LE. MIDPT-HWIDTH
                = YBASE for MIDPT-HWIDTH < X < MIDPT+HWIDTH
                = slp2*X+(YBASE-slp2*MIDPT-slp2*HWIDTH)
                        for MIDPT+HWIDTH .LE. X .LE. MIDPT+HWIDTH+(DEPTH+YOFF)/slp2
                = YBASE+DEPTH+YOFF      for X > MIDPT+HWIDTH+(DEPTH+YOFF)/slp2

usage summary
        * call as a function
        * generates hard eclipse light curve at specified points
```

```
subroutines
        NONE

example
        x=findgen(100)
        y=mk_lceclipse(x,60,20,0.77,pder,ybase=0.4,yoff=0.11,$
        angfall=178*!pi/180.,angrise=10.*!pi/180.) & plot,x,y

history
        vinay kashyap (MayMMVII; based on MK_LORENTZ)
        added keyword STEEP, also corrected bug of pderiv wrt ANG (VK; AugMMVII)
```

B.147 MK_LCOOL_APED [/scrypt]

```
script  MK_LCOOL_APED.PRO
        IDL program to set up the call to EXAPED and write out the
        emissivities from the linelist.fits file

usage
        wmn=0. & wmx=3000.
        atomdb='/data/fubar/SCAR/atomdb/atomdb_v2.0.2/' ;path to ATOMDB
        apeddir='/data/fubar/SCAR/emissivity/apedD'     ;output directory
        .run mk_lcool_aped

vinay kashyap (Jul02)
```

B.148 MK_LCOOL_CHIANTI [/scrypt]

```
script  MK_LCOOL_CHIANTI
        IDL program to call WRT_LN_CHIANTI repeatedly for various pressures
        and densities to generate a database of line intensities for all
        available CHIANTI ions

usage
        set the variables WMN and WMX to fix wavelength range in [Ang]
        set the variables NE_MIN, NE_MAX, D_NE [log10[cm^-3]] to set
        range of densities, and P_MIN, P_MAX, D_P [log10[cm^3 K]] to
        set range of pressures at which to compute the emissivities
        set CHIDIR as the path to the CHIANTI installation
        set OUTDIR as the path to where the outputs should be stored

        if initale has been run, CHIDIR is set to !CHIDIR, and OUTDIR
        is set to !TOPDIR/emissivity/chianti

        invoke as
        .run mk_lcool_chianti


vinay kashyap (Dec96)
  updated to make it more robust to other people's setups (VK; Jul13)
```

B.149 MK_LINSTR [/mkemis]

```
function        mk_linstr
        create a standard emissivity structure by putting together
        the various inputs.  this emissivity structure can then be
        used in PINTofALE exactly as though it had been read in via
        RD_LINE().

syntax
        linstr=mk_linstr(emis,logT=logT,wvl=wvl,Z=Z,ion=ion,jon=jon,$
        desig=desig,econf=econf,src=src,verbose=verbose)

parameters
        emis    [INPUT; required] emissivities
                * if 2D, assumed to be an array of size (NLOGT,NWVL)
                * if 1D, makes a semi-intelligent guess as to whether
                  it is an array corresponding to LOGT or WVL
                  -- first checks with LOGT and if size matches, that's it
                  -- else checks with WVL and
                  -- if that doesn't match then quits with an error

keywords
        logT    [INPUT] log10(Temperatures [K])
                * default is a regular grid going from 4..8
                * if any of the values are > 100, then it is assumed
                  that the input is in [K] and the log10 is taken
                * if size does not match first dimension of 2D EMIS,
                  then ignored and default is used
        wvl     [INPUT] wavelengths [Angstroms]
                * if size does not match 2nd dimension of EMIS, then
                  quits with an error
        Z       [INPUT] atomic numbers of elements generating given WVL
                * missing values will be set to 1
        ion     [INPUT] ionic state of element generating given WVL
                * missing values are set to 1
        jon     [INPUT] the ionic state that matters as far as ion-balance
                is concerned
                * missing values are set to ION
        src     [INPUT] reference to source of line info
                * e.g., 1=SPEX, 2=CHIANTI, 3=APED
                * if not set, assumed to be -99
                * if given but in insufficient numbers, the value of the
                  first element is extrapolated out
        desig   [INPUT] level designations for lower & upper level
        econf   [INPUT] e-configurations for lower & upper level
```

```
                * DESIG and ECONF are expected to be array of size (2,NWVL)
                  and are filled out with blank strings if parts are missing
        verbose [INPUT] controls chatter
        _extra  [JUNK] ignore -- here only to prevent crashing the program

history
        vinay kashyap (Jun02)
```

B.150 MK_LORENTZ [/fitting]

```
function        mk_lorentz
        returns a modified Lorentzian, Lb(X)

syntax
        l=mk_lorentz(x,mean,core,peak,pder,/betap,/normflx,missing=m,/fwhm)

parameters
        x       [INPUT array; required] where G(X) must be computed
        mean    [INPUT; default: mid_point(X)] position of peak
        core    [INPUT; default: 0.1*range(X)] core width
        peak    [INPUT; default: 1] value of Lb(X=MEAN)
        pder    [OUTPUT; optional] partial derivatives of model wrt
                parameters at each X; calculated only if 5 parameters
                are supplied in call.
                * array of size [N(X),4], with columns containing the partial
                  derivatives wrt MEAN, CORE, PEAK, and BETAP respectively

keywords
        betap   [INPUT; default=1] the index of the beta-profile.  default
                is regular Lorentzian.
                * if NORMFLX is set and BETAP.le.0.5, the default is used
        normflx [INPUT] if set, {\int_{-\infty}^{\infty} dX Lb(X) = PEAK}
        missing [INPUT] 3 element array to populate missing values of
                MEAN, CORE, and PEAK
        fwhm    [INPUT] if set, assumes that the input CORE is actually
                given as full-width at half-max, and converts to the true
                core width prior to calculation, and then converts back to
                full-width at half-max post-calc.
        _extra  [JUNK] here only to prevent crashing

description
        The Lorentzian is
                L(X) = PEAK / ( 1 + ((X-MEAN)/CORE)^2 )
        The modified Lorentzian is
                Lb(X) = PEAK / ( 1 + ((X-MEAN)/CORE)^2 ) ^ BETAP

        When integrated over the real axis (Gradshteyn & Ryzhik, 3.251,2),
                \int dX Lb = PEAK*CORE*B(1/2, BETAP-1/2),
        where B(x,y) is the Beta-function, with BETAP > 1/2

        hence, if NORMFLX is set,
                Lb(X) = (PEAK/CORE/B(1/2,beta-1/2))/(1+((X-MEAN)/CORE)^2)^BETAP
```

```
        The hwhm is defined as that value of X=W such that
                Lb(W)=0.5*Lb(X=MEAN)
        i.e.,   Lb(W)=(stuff)/(1+((W-MEAN)/CORE)^2)^(BETAP)=0.5*(stuff)
        hence,  hwhm = W-MEAN = CORE*sqrt(2^(1/BETAP)-1)
        or,     fwhm = CORE*2*sqrt(2^(1/BETAP)-1)

usage summary
        * call as a function
        * generates modified Lorentzian model only at specified points X
        * needs MEAN, CORE, PEAK for complete specification

examples
        x=findgen(1000)*0.01 & peasecolr & b=2.5
        L1=mk_lorentz(x,5,1,1) & L2=mk_lorentz(x,5,1,1,/fwhm)
        Lb1=mk_lorentz(x,5,1,1,betap=b) & Lb2=mk_lorentz(x,5,1,1,betap=b,/fwhm)
        Lf1=mk_lorentz(x,5,1,1,/normflx) & Lf2=mk_lorentz(x,5,1,1,betap=b,/norm)
        plot,x,L1 & oplot,x,L2,col=2
        oplot,x,Lb1,thick=2 & oplot,x,Lb2,thick=2,col=2
        oplot,x,Lf1,thick=2,line=1 & oplot,x,Lf2,thick=2,col=3,line=1

subroutines
        NONE

history
        vinay kashyap (Oct98)
        changed keyword BETA to BETAP; corrected normalization (VK; Dec98)
        recomputed partial derivatives (VK; MarMM)
        now works even if X are integers (VK; Jul01)
        what if PEAK=0? (VK; Apr02)
        converted array notation to IDL 5 (VK; Apr02)
        added partial derivative wrt BETAP to PDER output array (VK; Jun02)
        changed keyword NORM to NORMFLX (VK; Oct02)
        added keyword FWHM (VK; Apr03)
```

B.151 MK_MODEL [/fitting]

```
function        mk_model
        generate and return a compound model using the supplied
        parameters of components made up of 3-parameter functions

syntax
        model=mk_model(x,p,w,h,pder,group=group,delp=delp,type=type,/asis,$
        /allcomp,/fwhm,/norm,missing=missing,betap=betap,vrot=vrot)

parameters
        x       [INPUT; required] points where model is computed
        p       [INPUT; required] positions of components
        w       [INPUT; required] widths of components
        h       [INPUT; required] heights of components
        pder    [OUTPUT] contains 2D array (NX,3*NPAR) of partial derivatives
                of each component wrt parameters

keywords
        group   [INPUT] grouping index of gaussians
        delp    [INPUT] offsets from first position value in group
                NOTE: GROUP and DELP are ignored if ASIS is set
                NOTE: if GROUP and DELP are not given, ASIS is set
        type    [INPUT; default=’G*aussian’] what type of curve to generate
                for the model.  other possibilities are
                        L*orentzian
                        B*eta-profile (set "Beta=<value>")
                        S*lant (set "slant=<angle>,<beta>")
                        R*otation-convolved Gaussian (set "rot=<velocity>")
                        P*ower-law (set "pw=<break>" or "pe=<break>")
                        U*ser_defined (not yet implemented), etc.
                * if not defined, uses "Gauss"
                * if insufficiently defined, uses last defined as default
                  for the rest
        asis    [INPUT] the default action is to verify that the p are
                consistent with GROUP and DELP and to correct as necessary.
                if ASIS is set, this is not done.
        allcomp [INPUT] if set, returns the sum of components and also each
                component in a separate column of a 2D array of size
                (N_ELEMENTS(P)+1,N_ELEMENTS(X))
        _extra  [INPUT] pass defined variables to subroutine:-
                MK_GAUSS: MISSING, FWHM, NORM
                MK_LORENTZ: BETAP, MISSING, NORM
                MK_SLANT: ANGLE, BETAP, MISSING, NORM
                MK_ROGAUSS: VROT, MISSING, FWHM, NORM
```

```
usage summary
        * call as a function
        * returns composite model by adding up components
        * input component parameters as sequence of arrays of
          positions, widths, and heights
        * input grouping info by 2 additional arrays
        * specify type of model with keyword TYPE
        * partial derivatives are computed only if ALL 5 parameters are
          supplied in call
        * example TYPEs:
          'Gauss', 'gaussian', 'gau', 'g'
          'Lorentz', 'lorentz', 'lorentzian', 'lor', 'l'
          'beta=1.8', 'BETA=2.4', 'B=2', 'b 1.8', 'b:1.8', 'b   1.8'
          'slant=45,1.8','SLANT=89,2.4', 's=10','s=,2', 's:3,2'
          'rotational velocity=1e-4', 'rot=30', 'r:30e5', 'R 1e-4'
          'pe=1', 'pw=12.3985', 'pw@12.4'

subroutines
        MK_GAUSS
        MK_LORENTZ
        MK_SLANT
        MK_ROGAUSS
        MK_POWLAM
        KILROY

history
        vinay kashyap (Oct96)
        added parameter to return partial derivatives (VK; Nov96)
        added _EXTRA, added call to MK_LORENTZ (VK; Oct98)
        allowed setting BETAP for MK_LORENTZ via TYPE definition (VK; Dec98)
        added MK_SLANT calls (VK; MarMM)
        added MK_ROGAUSS, MK_POWLAM calls (VK; JunMM)
```

B.152 MK_POLY1D [/fitting]

```
function        mk_poly1d
        returns a polynomial, \Sum_{i=0}^{N} a_i (x-x0)^i

syntax
        py=mk_poly1d(x,aa,pder,x0=x0,verbose=verbose)

parameters
        x       [INPUT array; required] where G(X) must be computed
        aa      [INPUT; required] array of coefficients for the polynomial
                * the degree of the polynomial that is returned is N(AA)-1
        pder    [OUTPUT; optional] partial derivatives of model wrt parameters
                at each X; calculated only if 4 parameters are supplied in call.

keywords
        x0      [INPUT; default=0] zero-point of function
        verbose [INPUT; default=0] controls chatter
        _extra  [JUNK] here only to prevent crashing

usage summary
        * call as a function

subroutines
        NONE

history
        vinay kashyap (Aug01)
        converted array notation to IDL 5; summation of powers now kosher;
          corrected sign of partial wrt X0 (VK; Apr02)
```

B.153 MK_POWLAM [/fitting]

```
function        mk_powlam
        returns a power-law in WAVELENGTH G(X)

syntax
        g=mk_powlam(x,gamma1,gamma2,normE,pder,Xo=Xo,pbreak=pbreak,$
        NH=NH,wvlar=wvlar,effar=effar,exptime=exptime,dellam=dellam)

parameters
        X       [INPUT array; required] where G(X) must be computed
        gamma1  [INPUT; required] power-law index 1
        gamma2  [INPUT; required] broken power-law index 2
        normE   [INPUT; required] power-law normalization
                * as appropriate for an _energy [keV]_ scale
        pder    [OUTPUT; optional] partial derivatives of model wrt parameters
                at each X; calculated only if 5 parameters are supplied in call.
                * array of size [N(X),4], with columns containing the partial
                  derivatives wrt GAMMA1, GAMMA2, NORME, and PBREAK respectively

keywords
        Xo      [INPUT] Wavelength for which normalization
                is defined 1.0; default is 1.0 Angstroms
        pbreak  [INPUT] Wavelength at which power-law "breaks"
        NH      [INPUT] Hydrogen column density for computing ISM absorption
        wvlar   [INPUT array] wavelength grid of
                Chandra/HRC-S/LETG effective area
        effar   [INPUT array] Chandra/HRC-S/LETG effective area
        exptime [INPUT] exposure time
        dellam  [INPUT] wavelength binsize for computing COUNTS spectrum
        _extra  [JUNK]

description

  for X <= pbreak,
  G(X)=exp(-tau)*norm*(X/Xo)^(gamma1-2)*effar*exptime*dellam

  for X > pbreak,
  G(X)=exp(-tau)*norm*(pbreak/Xo)^(gamma1-gamma2)*(X/Xo)^(gamma2-2)
        *effar*exptime*dellam

usage summary
        * call as a function
        * generates model only at specified points X
        * needs for complete specification
```

```
subroutines
        ISMTAU

history
        Deron Pease (June 2000)
        call ISMTAU only if NH is defined (VK; MMJul)
        bug correction: if NH not defined, etau must be set to 1 (VK; Jun01)
        converted array notation to IDL 5 (VK; Apr02)
        added partial derivatives wrt PBREAK (VK; Jun02)
        pushed partial derivatives wrt PBREAK inside if..then (VK; Nov02)
```

B.154 MK_RECIPROCALPOW [/fitting]

```
function        mk_reciprocalpow
        returns a reciprocal double Power-law function,
          p(X)  = 1/(A1*X^GAMMA1 + A2*X^GAMMA2),
          with A1*J^GAMMA1 = A2*J^GAMMA2, so A2 == A1*J^(GAMMA1-GAMMA2)
          and A1 == NORM
          for X>0

syntax
        p=mk_reciprocalpow(x,norm,gamma1,gamma2,pjoin,pder,verbose=verbose)

parameters
        X       [INPUT; required] where p(X) must be computed
        norm    [INPUT; required] normalization
        gamma1  [INPUT; required] power-law index 1
        gamma2  [INPUT; required] power-law index 2
        pjoin   [INPUT; required] the joining point where the two
                power-law components are equal
        pder    [OUTPUT; optional] partial derivatives of model wrt parameters
                at each X; calculated only if 6 parameters are supplied in call.
                * array of size [N(X),4], with columns containing the partial
                  derivatives wrt NORM, GAMMA1, GAMMA2, and PJOIN respectively

keywords
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

example
        x=(findgen(100)+1)/10. & p=mk_reciprocalpow(x,1.,1.5,0.2,2) & plot,x,p,/xl,/yl
        contrast with: oplot,x,mk_bknpower(x,0.3,-1/1.5,-5,2),line=2

history
        Vinay Kashyap (Sep2008)
```

B.155 MK_ROGAUSS [/fitting]

```
function        mk_rogauss
        convolves a gaussian line profile with a rotational broadening function
        and returns the broadened profile.

syntax
        rg=mk_rogauss(x,mean,sig,peak,pder,vrot=vrot,/fwhm,/normflx,$
        missing=missing)

parameters
        x       [INPUT array; required] where rG(X) must be computed
                * generally in a wavelength grid
        mean    [INPUT] passed w/o comment to MK_GAUSS
        sig     [INPUT] passed w/o comment to MK_GAUSS
        peak    [INPUT] passed w/o comment to MK_GAUSS
        pder    [OUTPUT; optional] partial derivatives of model wrt
                parameters at each X
                * calculated only if 5 parameters are supplied in call.
                * numerically computed because of intractable convolution
                * array of size [N(X),4], with columns containing the partial
                  derivatives wrt MEAN, SIG, PEAK, and VROT respectively

keywords
        vrot    [INPUT; default=0] the rotational velocity, as a fraction
                of light speed
                * if > 1 and < 3e5, assumed to be in [km/s]
                * if > 3e5 and < 3e10, assumed to be in [cm/s]
                * if -ve, then convolves with a fixed rotation profile
                  whose width is defined with reference to MEAN
        _extra  [JUNK] pass defined keywords to MK_GAUSS
                FWHM, NORMFLX, MISSING

description
        NORMFLX=0 && FWHM=0:
          G(X)=PEAK*exp((X-MEAN)^2/SIGMA^2)
        NORMFLX=1 && FWHM=0:
          G(X)=(PEAK/SIGMA/sqrt(2*!PI))*exp((X-MEAN)^2/2/SIGMA^2)
        NORMFLX=0 && FWHM=1:
          G(X)=PEAK*exp(2.355^2*(X-MEAN)^2/2/SIGMA^2)
        NORMFLX=1 && FWHM=1:
          G(X)=(2.355*PEAK/SIGMA/sqrt(2*!PI))*exp(2.355^2*(X-MEAN)^2/2/SIGMA^2)

        G(X) is then convolved with
          V(X';X)=(2/!PI/HW)*sqrt(1.-((X'-X)/HW)^2), where HW=VROT*X
```

```
        (note that V(X';X) *changes* across the range of X)

usage summary
        * call as a function
        * needs MEAN, SIG, PEAK for complete specification

subroutines
        MK_GAUSS
        also makes recursive call to itself if PDER is required

history
        vinay kashyap (JunMM, based on MK_SLANT)
        converted array notation to IDL 5 (VK; Apr02)
        added partial wrt VROT to PDER (VK; Jun02)
        changed keyword NORM to NORMFLX (VK; Oct02)
```

B.156 MK_SINUSOID [/fitting]

```
function        mk_sinusoid
        returns a sinusoid, A*sin(f*X+p)+c

syntax
        s=mk_sinusoid(x,ampl,freq,offset,pder,phase=phase,/normflx,$
        missing=missing)

parameters
        x       [INPUT array; required] where G(X) must be computed
        ampl    [INPUT; default: 1] amplitude of the sinusoid
        freq    [INPUT; default: 2*!pi/range(X)] frequency of sinusoid
        offset  [INPUT; default: 1] constant offset for sinusoid
        pder    [OUTPUT; optional] partial derivatives of model wrt
                parameters at each X; calculated only if 5 parameters
                are supplied in call.
                * array of size [N(X),4], with columns containing the partial
                  derivatives wrt AMPL, FREQ, OFFSET, and PHASE respectively

keywords
        phase   [INPUT; default=0] the phase of the sinusoid in [deg]
        normflx [INPUT] not used yet
        missing [INPUT] 3 element array to populate missing values of
                AMPL, FREQ, and OFFSET
        _extra  [JUNK] here only to prevent crashing

description
        The sinusoid is
                S(X) = AMPL * sin(FREQ*X+PHASE) + OFFSET

usage summary
        * call as a function
        * generates sinusoid values only at specified points X
        * needs AMPL, FREQ, OFFSET, and PHASE for complete specification

examples

subroutines
        NONE

history
        vinay kashyap (Sep04)
```

B.157 MK_SLANT [/fitting]

```
function        mk_slant
        returns the product of a slanted line with a modified
        Lorentzian, Lb(X)

        the idea is to model the asymmetrical line profiles seen in
        CXO grating data.

syntax
        Lbs=mk_slant(x,mean,core,peak,pder,angle=angle,betap=betap,$
        /normflx,missing=missing)

parameters
        x       [INPUT array; required] where G(X) must be computed
        mean    [INPUT; default: mid_point(X)] position of peak
        core    [INPUT; default: 0.1*range(X)] core width
        peak    [INPUT; default: 1] value of Lb(X=MEAN)
        pder    [OUTPUT; optional] partial derivatives of model wrt parameters
                at each X; calculated only if 5 parameters are supplied in call.
                * array of size [N(X),5], with columns containing the partial
                  derivatives wrt MEAN, CORE, PEAK, BETAP, and ANGLE respectively

keywords
        angle   [INPUT; default=0] the atan(slope) of the straight line
                that makes the Lorentzian asymmetric
        betap   [INPUT; default=1] the index of the beta-profile.  default
                is regular Lorentzian.
                * if NORMFLX is set and BETAP.le.0.5, the default is used
        normflx [INPUT] if set, {\int_{-\infty}^{\infty} dX Lbs(X) = PEAK}
        missing [INPUT] 3 element array to populate missing values of
                MEAN, CORE, and PEAK
        _extra  [JUNK] here only to prevent crashing

description
        The Lorentzian is
                L(X) = PEAK / ( 1 + ((X-MEAN)/CORE)^2 )
        The modified Lorentzian is
                Lb(X) = PEAK / ( 1 + ((X-MEAN)/CORE)^2 ) ^ BETAP
        The asymmetric modified Lorentzian is
                Lbs(X) = (PEAK + PEAK*SLOPE*(X-MEAN)) / $
                        ( 1 + ((X-MEAN)/CORE)^2 ) ^ BETAP

        When integrated over the real axis (Gradshteyn & Ryzhik, 3.251,2),
        the second terms drops out and the integral is identical to that
```

```
        of the modified Lorentzian itself, i.e.,
                \int dX Lbs = CORE*PEAK*B(1/2,BETAP-1/2)
        where B(x,y) is the Beta-function, with BETAP > 1/2

        if NORMFLX is set,
                Lbs(X) = ((PEAK+PEAK*SLOPE*(X-MEAN))/CORE/B(1/2,beta-1/2)) / $
                         (1+((X-MEAN)/CORE)^2)^BETAP

usage summary
        * call as a function
        * generates modified Lorentzian model only at specified points X
        * needs MEAN, CORE, PEAK for complete specification

subroutines
        NONE

history
        vinay kashyap (MarMM, based on MK_LORENTZ)
        now works even if X are integers (VK; Jul01)
        converted array notation to IDL 5 (VK; Apr02)
        accounted for case PEAK=0, added partial derivatives of BETAP and
          ANGLE to PDER (VK; Jun02)
        changed keyword NORM to NORMFLX (VK; Oct02)
```

B.158 MODALPOINT [/stat]

```
function modalpoint
        returns the mode of distribution defined by set of unbinned array values

syntax
        arraymode=modalpoint(array,eps=eps,verbose=verbose)

parameters
        array   [INPUT; required] array of values for which mode must be found

keywords
        eps     [INPUT; default=1e-6] a small number
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

description
        sort the array, divide into two sets around midpoint of range,
        choose the set that contains more elements, and repeat until
        the range is sufficiently small, and declare the midpoint of
        the range to be the mode

warning
        may fail for multimodal distributions

example
        for i=0,20 do print,i,modalpoint(randomn(seed,10000L)+i)

history
        translated to IDL by Vinay Kashyap from C code written for BEHR
          by Taeyoung Park c.2003 (MarMMVI)
        now correctly handles case when input is quantized (VK; SepMMVI)
```

B.159 MORFO_CIRCOLARE [/util]

```
function        morfo_circolare
        returns an image that is morphologically manipulated with a
        rotated, annular arclike structure element

        if MOPEN is set then an opening operation is applied
        if MCLOSE is set then a closing operation is applied

syntax
       result=morfo_circolare(image,arc,rzero,rstep,nstep,gray=gray,$
       /mopen,/mclose,verbose=verbose)

parameters
        image   [INPUT; required] the image you want to smooth
                * must be a 2-D array
        arc     [INPUT; required] the angular size of the arc to be used
                as a structuring element [degree]
                * the full circle is completed by moving the arc by ARC/2
                  and combining the results
        rzero   [INPUT; required] radius of the innermost annulus to consider
        rstep   [INPUT; required] the annular width of structure elements [pix]
        nstep   [INPUT] number of annuli to use
                * if not given, carries on until the outer radius equals
                  the larger of the image width or height

keywords
        gray    [INPUT] explicitly set to 0 to carry out the morphological
                operations in binary rather than grayscale; default is to
                use grayscale
       mopen    [INPUT] if set, runs the IDL function morph_open()
       mclose   [INPUT] if set, runs the IDL function morph_close()
                * NOTE: if more than one of MOPEN,MCLOSE
                  are set, the operations will be daisy-chained in that
                  particular order
        readd   [INPUT] if set, adds up the output from the morphological
                operations at one annular arc to the output from another
                * the default is to record, at any pixel, the largest
                  response obtained from all the bits
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (july 06; based on morfo_rottangoli)
        added keyword _EXTRA (VK; may 07)
```

B.160 MORFO_POTARE [/util]

```
function        morfo_potare
        prunes the little tributaries from a morphological skeleton
        of an image and returns a cleaner image of the skeleton.

        this pruning works as follows: first, branch points in the
        skeleton are identified, and they and their neighboring points
        are extracted and set aside; next, the remaining points are
        segmented into distinct and separate lines and then all segments
        which are smaller than some threshold are discarded; and finally
        the branch points and their neighborhoods are added back on.

        this is not the standard algorithm for pruning, as far as I know.

syntax
        pruned_skeleton=morfo_potare(skel,minpix,verbose=verbose)

parameters
        skel    [INPUT; required] the input 2D bitmap image that
                contains a morphological skeleton
                * NOTE: input _must_ be a skeleton, otherwise the
                  program will fail miserably
        minpix  [INPUT] minimum number of pixels that must exist
                in a given branch for it to be kept on
                * default is 4

keywords
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

subroutines
        MORFO_SEGMENTO

history
        vinay kashyap (Jul2006)
        added _EXTRA keyword; changed call from MORFO_FILTRI to
          MORFO_SEGMENTO (VK; May2007)
```

B.161 MORFO_ROTTANGOLI [/util]

```
function        morfo_rottangoli
        an image that is morphologically manipulated with a
        rotated, rectangular structure element

        if MGRAD is set then a gradien operation is applied
        if MTOP is set then a tophat operation is applied
        if MOPEN is set then an opening operation is applied
        if MCLOSE is set then a closing operation is applied
        if MTHIN is set then a thinning operation is applied

syntax
       result=morfo_rottangoli(image,xsize,ysize,angle,gray=gray,$
       /mgrad,/mtop,/mopen,/mclose,/mthin,/readd,/reuse,$
        verbose=verbose)

parameters
        image   [INPUT; required] the image you want to smooth
                * must be a 2-D array
        xsize   [INPUT; required] the width of the structure element in pixels
        ysize   [INPUT; required] the height of the structure element in pixels
        angle   [INPUT; default=0] the angle in degrees by which to rotate the
                rectangular structure element defined by (XSIZE,YSIZE)
                * if a vector, the output will be the summed images
                  from the application of the rectangular structure
                  element at each of the angles

keywords
        gray    [INPUT] explicitly set to 0 to carry out the morphological
                operations in binary rather than grayscale; default is to
                use grayscale
       mgrad    [INPUT] if set, runs the IDL function morph_grad()
       mtop     [INPUT] if set, runs the IDL function morph_tophat()
       mopen    [INPUT] if set, runs the IDL function morph_open()
       mclose   [INPUT] if set, runs the IDL function morph_close()
       mthin    [INPUT] if set, runs the IDL function morph_thin()
                * NOTE: if more than one of MGRAD,MTOP,MOPEN,MCLOSE,MTHIN
                  are set, the operations will be daisy-chained in that
                  particular order
        readd   [INPUT] if set, adds up the output from the morphological
                operations at one ANGLE to the output from another ANGLE
                * the default is to record, at any pixel, the largest
                  response obtained from all the ANGLEs
        reuse   [INPUT] if set, uses the output from the morphological
```

```
                operations at one ANGLE as the input for the next ANGLE
                * overrides both default and READD
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

history
        julia sandell (july 06)
        tightened up and made bulletproof and changed name from morph_rot
          to morfo_rottangoli; morphological analysis with rotating rectangles;
          added keywords GRAY, READD, REUSE, VERBOSE (Vinay Kashyap; july 06)
        added _EXTRA keyword (VK; may 07)
```

B.162 MORFO_SCHELETRO [/util]

```
function        morfo_scheletro
        compute and return the skeleton representation of an
        image A using a circular structuring element B.
        a skeleton is defined as the union of the sets
                Erosion(A,k*B) - Opening(Erosion(A,k*B),B)
        for k=0,..K where K is the largest value of k before
        the sets become empty.

syntax
        skel=morfo_scheletro(image,rmin,thresh=thresh,verbose=verbose)

parameters
        image   [INPUT; required] image whose skeleton must be
                obtained.
                * must be a 2-D array
        rmin    [INPUT] radius of the circle that
                acts as a structuring element
                * if not given, the default is to use a 3x3 array
                * if given, uses DIST() to define the array
                  -- minimum possible radius is hardcoded as 1pix

keywords
        thresh  [INPUT; default=0] filter IMAGE such that only
                values greater than THRESH are considered.
                * if size matches that of IMAGE, a pixel-by-pixel
                  filtering is carried out
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (Jul2006)
```

B.163 MORFO_SEGMENTO [/util]

```
function        morfo_segmento
        percolates along adjacent connected pixels in a bitmap
        until all the pixels that are connected to each other get
        grouped into contiguous regions and are so labeled; an image
        containing the region numbers for each pixel is returned

syntax
        oimg=morfo_segmento(img,areas,thresh=thresh,arclev=arclev,$
        subidx=subidx,/hardcut,deepimg=deepimg,verbose=verbose)

parameters
        image   [INPUT; required] 2D image in which to separate
                out the connected blobs
                * the blobs are defined as any pixel with intensity>0
                  unless keyword THRESH is set
        areas   [OUTPUT] number of pixels in each labeled region
                * remember the IDL index shift! AREAS[0] is the number
                  of pixels in the background, and AREAS[SOURCE] gives
                  the number of pixels that constitute the blob SOURCE

keywords
        thresh  [INPUT] the threshold value at which to convert
                the input image into a bitmap
                * if not set, assumed to be 0
        arclev  [INPUT] fraction of the maximum area size to keep
                in the output -- all regions of size smaller than
                this are zeroed out
                * default is 0, i.e., no filtering is done
                * if .le. -1, taken to be an absolute value cut
                  if .gt. -1 && .lt. 0, absolute value is choosen
                  if .ge. 1 && .lt. 100, assumed to be percentage
                  if .ge. 100, assumed to be 1-1/ARCLEV
        subidx  [INPUT] index array defining a subset of IMAGE
                to which one must confine attention
                * if not set, assumed to be LINDGEN(N_ELEMENTS(IMAGE))
        hardcut [INPUT] if set, places a hard cut at the boundaries
                defined by SUBIDX -- the default is to track the region
                out of SUBIDX if any part of it overlaps
        deepimg [OUTPUT] an image that depicts the depth at which a
                particular pixel was added to a blob
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program
```

```
description
        this is an updated version of morfo_filtri(), and is much faster
        because it uses a few algorithmic and IDL tricks to speed things
        up.  for example, rather than do a linear search going through
        every pixel once, it does array operations.  each pixel is initially
        assigned a unique number, and at each iteration, the numbers of
        each adjacent pixels are compared and the smallest of the numbers
        is assigned to all of them.  thus, all the pixels that belong to
        each blob end up having the same number.  these numbers are then
        recast into a gap-free sequence thereafter.
        note: there will be no significant improvements in speed in some
        special cases, such as if there is a very large connected region;
        this is never *slower* than morfo_filtri() however.

subroutines
        KILROY

history
        vinay kashyap (Sep2006; based on morfo_filtri.pro)
        added keywords ARCLEV and _EXTRA (VK; May2007)
        was failing when there was only one blob (VK; Jun2007)
```

B.164 MORFO_SOGLIA [/util]

```
function        morfo_soglia
        threshold the image based on some cut derived from applying some
        kind of filter, and return the image, ideally with the interesting
        features enhanced

syntax
        fimg=morfo_soglia(img,gamma=gamma,nsigma=nsigma,centrale=centrale,$
        bitlev=bitlev,/zeromin,mimg=mimg,thresh=thresh,verbose=verbose)

parametes
        img     [INPUT; required] input image to be filtered
                * must be 2D

keywords
        snr     [INPUT] if given, computes a threshold based
                on a S/N criterion
                * if set, IMG is _not_ converted to a byte scale
        gamma   [INPUT] if given and is +ve, calls GMASCL()
                to transform the image first according to
                the power-law scaling transformation
        nsigma  [INPUT] if given, sets the threshold at
                mean+NSIGMA*stddev
                * default is 1
                * if mean+NSIGMA*stddev < 0, this is not applied
        centrale [INPUT] if set, carries out a local median
                filtering with a box of size 2*fix(CENTRALE>1)+1
                * NSIGMA thresholding is ignored unless NSIGMA
                  is explicitly set
                * the filtered pixels are zero'd out unless the keyword
                  ZEROMIN is explicitly set to 0
        bitlev  [INPUT] if set, carries out a bit-level filtering
                down to level BITLEV, with 1=most significant and
                7 or more=least significant
                * both CENTRALE and NSIGMA are ignored
        zeromin [INPUT] if set, sets all pixels that do not pass
                the threshold to 0; otherwise will be set to the
                threshold value
        mimg    [OUTPUT] the median image, constructed if CENTRALE is set
        thresh  [OUTPUT] the threshold value used, if histogram-threshold
                is applied
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program
```

```
restrictions
        requires David Fanning's GMASCL() function

history
        vinay kashyap (Jul2006)
        added _EXTRA keyword; bugfix when NSIGMA was very small (VK; May2007)
```

B.165 MUNGE_LIST [/]

```
procedure       munge_list
        chomps through ascii file containing observed wavelengths/fluxes
        and possible matches to them, then generates the appropriate
        line database and ID structures (see RD_LINE and LINEID)

syntax
        munge_list,list,idstr,lnstr,wvl,flux,fluxerr,dbdir=dbdir,$
        sep=sep,prefix=prefix,comment=comment, mapping=mapping,pres=pres,$
        logp=logp,n_e=n_e,desig=desig,econf=econf,chifil=chifil,chidir=chidir,$
        eqfile=eqfile,DEM=DEM,abund=abund,/noph,effar=effar,wvlar=wvlar

parameters
        list    [INPUT; required] file containing wavelengths, fluxes, and
                matches.
                * see format below
        idstr   [OUTPUT] LINEID type structure
        lnstr   [OUTPUT] RD_LINE type structure
        wvl     [OUTPUT] all the observed wavelengths
        flux    [OUTPUT] all the measured wavelengths
        fluxerr [OUTPUT] errors on FLUX

keywords
        dbdir   [I/O] directory in which to look for files
                * will be used ONLY if not given in LNLIST
                * if not specified, *default* will be set at run-time to
                  whatever was used last
                * on output, will contain complete list of line database
                  directories used
        sep     [INPUT; default=<tab>] separator used to delineate fields
                in LNLIST
        prefix  [INPUT; default=’*’] any line beginning with this letter
                is considered a comment
                * this is in addition to "%", "#", ";", "/"
        comment [OUTPUT] any commented lines are returned in this variable
                as a string array
        _extra  [INPUT ONLY] use this to pass defined keywords to
                -- RD_LIST (MAPPING)
                -- RD_LINE (PRES,LOGP,N_E,DESIG,ECONF,VERBOSE)
                -- FOLD_IONEQ (CHIFIL,CHIDIR,EQFILE,VERBOSE)
                -- LINEFLX (DEM,ABUND,NOPH,EFFAR,WVLAR)

input file format
        * one entry per line
```

```
        * each observed wavelength and related IDs must be in a block
          of the form
                /{ <sep> WVL <sep> FLUX <sep> FLUXERR
                ELEM1 ION1 <sep> WAVE1 <sep> SOURCE1 <sep> DESCRIPTION1
                ELEM2 ION2 <sep> WAVE2 <sep> SOURCE2 <sep> DESCRIPTION2
                ...
                ELEMn IONn <sep> WAVEn <sep> SOURCEn <sep> DESCRIPTIONn
                // human readable comments
                /}
        * the beginning and ending "/{" and "/}" are REQUIRED
        * WVL, FLUX, and FLUXERR are optional, and will be guessed at from
          the IDs:
          WVL is set to 0 or the WAVE of the strongest line;
          FLUX is the total of the peak emissivities of the IDs;
          FLUXERR is 1% of FLUX (if non-zero), or 1 otherwise.
        * obviously, WVL is required if FLUX is given, and WVL and FLUX
          are required if FLUXERR is given.
        * of the IDs, ELEM is required, ION is useful, WAVE and SOURCE are
          optional (obviously, WAVE is required if SOURCE is given, and
          WAVE and SOURCE are required if DESCRIPTION is given.)
        * due to a shortcoming in ZION2SYMB, it is best if "ELEM ION" is
          given as "ELEMENT ION", e.g., "Fe XII" or "FeXII" or "Fe 12"
          and not "26 12" or "26_12"
        * <SEP> is a separator, usually a <tab>, but had better not be any
          of "," "+-" "-" and "?"
        * WAVE may be (any combination of):-
          exact match           -- "WVL"
          inexact match         -- "WVL +- dW" or "WVL ? dW"
          range                 -- "WMIN,WMAX" or "WMIN-WMAX"
          enclosed in brackets  -- "()" "[]" "{}"
        * lines prefixed by PREFIX, "%", "#", ";" and "/" are ignored
          (except for "/{", "/}", and "//" -- the last is treated as
          scribbles to go into the NOTES field of the ID)
        * if no IDs are found, WVL is flagged as "UNKNOWN"
        * emissivities of IDs not attached to any WVL are returned in LNSTR

subroutines
        RD_LIST
          RD_LINE
            SYMB2ZION
              LAT2ARAB
          FOLD_IONEQ
            RD_IONEQ
              READ_IONEQ
          CAT_LN
```

```
        LINEFLX
          GETABUND
          WHEE
        IS_KEYWORD_SET

history
        vinay kashyap (Nov98)
        updated to include FLUXERR in IDSTR (VK; Dec98?)
        updated to include NOTES in IDSTR (VK; Mar99)
        updated for IDL5.6 keyword_set([0]) behavior change for vectors
          (VK; 20Mar2006)
        corrected bug where LNSTR was being concatenated if defined
          on input; now input is ignored (VK; Jul2008)
        corrected bug where LNSTR was not being output (VK; Jan2015)
```

B.166 NARY2DEC [/util]

```
procedure       nary2dec
        converts num from n-ary base to decimal

syntax
        nary2dec,num,dec,/binary,/octal,/hex

parameters
        num     [INPUT] n-ary number or string
        dec     [OUTPUT] corresponding decimal number

keywords
        binary  [INPUT] if set, NUM taken to be in binary notation
        octal   [INPUT] if set, NUM taken to be in octal notation
        hex     [INPUT] if set, NUM taken to be in hexadecimal notation
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (1996?)
```

B.167 NENH [/]

```
function        nenH
        calculate the ratio N(e)/N(H) (or N(e)/N(p) -- see keyword
        NPROTON below) in a plasma of specified temperature

syntax
        r=nenh(logT,abund=abund,eqfile=eqfile,chidir=chidir,elem=elem,$
        xelem=xelem,/nproton)

parameters
        logT    [INPUT; required] temperature(s) (log_10(T [K])) at which
                to compute ratio.

keywords
        abund   [I/O] abundances of elements.  default is Allen abundances.
        eqfile  [INPUT] pathname, relative to CHIDIR, of file containing
                ionization equilibrium values, passed w/o comment to RD_IONEQ
        chidir  [INPUT] path name to the CHIANTI installation, passed w/o
                comment to RD_IONEQ
        elem    [INPUT] return data for only these elements (default: all)
        xelem   [INPUT] exclude these elements (overrides ELEM)
                * default: none
                  (note that older CHIANTI ion balance files do not have
                  Cu, Zn, so originally XELEM was set by default to [29,30])
        nproton [INPUT] if set, returns the ratio N(e)/N(p)
        _extra  [JUNK] ignore -- here only to prevent crashing program

subroutines
        GETABUND [RDABUND]
        RD_IONEQ [READ_IONEQ (a CHIANTI routine)]
        SYMB2ZION [LAT2ARAB]
        KILROY

history
        vinay kashyap (Sep97)
        modified to have just one call to RD_IONEQ instead of one for
          each Z, changed default of XELEM to none (VK; Jun02)
```

B.168 NOISMOOTH [/util]

```
function        noismooth
        return the input function boxcar smoothed at varying
        scales such that the value at any point is determined
        at a sufficiently high S/N.

        beware that this will NOT conserve flux, and the adjacent
        points (or errors) of the smoothed function will no longer
        be statistically independent.

syntax
        ys=noismooth(y,ysig=ysig,snrthr=thresh,snrout=snr,xsiz=xsiz,$
        sdepth=sdepth,verbose=v)

parameters
        y       [INPUT; required] array that must be smoothed
                * note that integer count arrays are returned as
                  integers, which may not necessarily be what is
                  optimal.  to return floats instead, simply input
                  (y*1.0)

keywords
        ysig    [INPUT] errors on Y -- S/N on smoothed function will
                be calculated by square-adding
                * if not given, Gehrel's approximation to Poisson,
                  sqrt(abs(Y)+0.75)+1 will be assumed
                * if single-element, then assumed to be
                  -- additive constant if <0 (i.e., abs(YSIG[0]), or else
                  -- fractional if <1, or else
                  -- percentage if <100, and
                  -- additive constant otherwise
        snrthr  [INPUT] S/N threshold -- try to make every point lie
                above this S/N
                * default is 3
        snrout  [OUTPUT] final S/N at each point
        xsiz    [OUTPUT] final smoothing scale at each point
        sdepth  [INPUT] maximum smoothing scale to consider
                * default is n_elements(Y)/2-1, which is also the
                  hardcoded maximum
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

description
        akin to Harald Ebeling's adaptive smoothing of 2-D images
```

```
        (encoded in CIAO CSMOOTH), this program boxcar smooths the
        input 1-D array over a number of scales and at each bin
        keeps the result from the smallest scale which puts it
        over the specified S/N threshold.

examples
        peasecolr & y=randomu(seed,1000,poisson=1) & y[400:499]=randomu(seed,100,poisson
        plot,y & oplot,smooth(y,5),col=2,thick=3 & oplot,smooth(y,100),col=29,thick=3
        ys=noismooth(y,ysig=ysig,snrthr=10,snrout=snr,xsiz=xsiz)
        plot,y & oplot,ys,col=2,thick=3 & oplot,snr,col=3
        ys=noismooth(y,ysig=ysig,snrthr=3,snrout=snr,xsiz=xsiz,verbose=10)
        oplot,ys,col=25,thick=3 & oplot,snr,col=35
        ys=noismooth(y,ysig=ysig,snrthr=13,snrout=snr,xsiz=xsiz)
        oplot,ys,col=29,thick=3 & oplot,snr,col=39

these are all the smoothing tools in PINTofALE
        ALALOESS() makes a loess curve
        CLTSMOOTH() removes -ves from background subtracted spectra or light curves
        CONV_RMF convolves with specified RMF
        HAARTRAN() smoothes by filtering on Haar wavelet coefficients
        LINEREM() removes lines from spectrum by iterative S/N filtering
        NOISMOOTH() does boxcar accumulation a la Ebeling's asmooth
        REGROUP() accumulates from one end
        SCRMF does fast convolution using optimized RMF
        SMOOTHIE does peak-descent accumulation to build up S/N
        SPLAC computes a segmented piecewise linear approximations
        UNKINK() removes sharp kinks from a curve
        VARSMOOTH() does local smoothing at specified scales
        VOORSMOOTH() does fast local smoothing at specified scales

history
        vinay kashyap (OctMM)
        modified YSIG behavior to handle YSIG[0]>1 (VK; Nov'02)
```

B.169 NRG2CHAN [/util]

```
procedure       nrg2chan
        convolves input energy spectrum with an OGIP-style response matrix

syntax
        nrg2chan,nrg,flx,chan,spec,rmstr,effar=effar,nrgar=nrgar

parameters
        nrg     [INPUT; required] bin boundary values at which input FLX is
                defined.
                * usually [keV], depends on RMF file
        flx     [INPUT; required] fluxes in [ph/bin/(s/cm^2/sr/...)]
        chan    [OUTPUT; required] bin boundary values at which output SPEC
                is defined
                * same units as NRG
        spec    [OUTPUT; required] output, convolved spectrum
        rmstr   [INPUT; required] structure containing all the relevant
                data (e.g., see output of RD_OGIP_RMF):
                {NNRG,ELO,EHI,NCHAN,EMN,EMX,N_GRP,F_CHAN,N_CHAN,MATRIX}
                * ELO and EHI refer to range of photon energies at which
                  response is valid
                * EMN and EMX refer to bin boundaries for each channel
                * N_GRP, F_CHAN, and N_CHAN refers to number of groups
                  of non-zero data, beginning channel, and number of
                  channels in each group

keywords
        effar   [INPUT] effective areas [cm^2]
        nrgar   [INPUT] energies at which EFFAR are defined
                * same units as NRG
                * sizes of EFFAR and NRGAR must match
                * range of NRGAR must overlap NRG
                * if EFFAR and NRGAR are legal, FLX is multiplied by
                  EFFAR (over intersection of NRGAR and NRG; zeroed
                  elsewhere) before convolving with response matrix
                * of course, this multiplication makes sense only if
                  units on FLX are [ph/bin/cm^2/...]
        _extra  [JUNK] ignore - here only to prevent crashing

restrictions
        requires subroutine BINERSP
        requires output of RD_OGIP_RMF

history
```

```
vinay kashyap (Apr99; based on CONV_RMF)
```

B.170 NUM2STR [/util]

```
FUNCTION        num2str
        given an array of integers, prepends the appropriate number
        of zeros to make all the strings of the same length, equal
        to the maximum length needed.

SYNTAX
        sss=num2str(nnn,len=len,nmax=nmax)

PARAMETERS
        num     [INPUT; required] array of integers

KEYWORDS
        len     [INPUT] if set, final string length = LEN.
                * if LEN < maximum length, defaults to maximum
                * if LEN < 0, overrides above setting
        nmax    [INPUT] the maximum possible value of NUM
                * if set, computes LEN on the fly based on NUM
                * if both LEN and NMAX are set, LEN takes precedence
        _extra  [JUNK] here only to prevent crashing the program

EXAMPLE
        print, num2str(findgen(10),len=-4)

HISTORY
        written by vinay kashyap (1996?)
        added keyword NMAX (VK; Jan2000)
        now takes advantage of the In.n format (VK; Jul2005)
```

B.171 OGIPZIP [/util]

```
procedure       ogipzip
        compress a numerical array to return only those elements
        which are above some threshold, and the means to reconstruct
        the full fledged array.  basically this is how OGIP RMF matrix
        values are compressed and stored.

syntax
        ogipzip,arr,carr,ngrp,fchan,nchan,eps=eps,chan0=chan0,cchan=cchan

parameters
        arr     [INPUT; required] input 1D array to be compressed
        carr    [OUTPUT; required] compressed array
        ngrp    [OUTPUT; required] number of groups into which ARR is broken
        fchan   [OUTPUT; required] indices of first elements of each group
        nchan   [OUTPUT; required] number of elements in each group

keywords
        eps     [INPUT] a small number, below which to ignore all values
                * default is 1e-6
        chan0   [INPUT] index number of first channel, to determine
                whether FCHAN should be 0-based or 1-based
                * default is 1-based, because it appears that that is
                  how Chandra RMFs are being made.  set CHAN0 explicitly
                  to 0 to make it 0-based.
        cchan   [OUTPUT] channels corresponding to CARR
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (Nov2001)
```

B.172 OS_FAMILY [/external]

```
Project     :  SOHO - CDS

Name        :  OS_FAMILY()

Purpose     :  Return current operating system as in !VERSION.OS_FAMILY

Category    :  Utilities, Operating_system

Explanation :  Return the current operating system as in !VERSION.OS_FAMILY

       OS_FAMILY is assumed to be 'unix' if !VERSION.OS is not 'windows',
               'MacOS' or 'vms'

       To make procedures from IDL V4.0 and later compatibile with earlier
       versions of IDL, replace calls to !VERSION.OS_FAMILY with OS_FAMILY().

       Can also be used to replace calls to !VERSION.OS if care is taken with
       the change of case between 'Windows', which is what this routine
       returns, and 'windows' which is what !VERSION.OS returned in versions
       of IDL prior to 4.0.

Syntax      :  Result = OS_FAMILY()

Examples    :  IF OS_FAMILY() EQ 'vms' THEN ...

Inputs      :  None.

Outputs     :  The result of the function is a scalar string containing one of
               the four values 'Windows','MacOS','vms' or 'unix'

Opt. Outputs:  None.

Keywords    :  LOWER - set to return lowercase strings
              NAME - return OS_NAME (if available)

Prev. Hist. :  Written,  W. Landsman

History     :  Version 1, 29-Aug-1995, William Thompson, GSFC
                       Incorporated into CDS library
              Version 2, 15 May 2000, Zarro (SM&A/GSFC) - added /LOWER

              Version 3, 22-Dec-2002, Zarro (EER/GSFC) - saved result in
                       common block for speed
```

```
                Modifed, 2-June-2006, Zarro (L-3Com/GSFC) - removed
                          COMMON and added NAME keyword

Contact      :  WTHOMPSON
```

B.173 PEASECOLR [/util]

```
procedure       peasecolr
        set up a color table by loading in some nice colors

syntax
        peasecolr,/white,/brewer,verbose=verbose,/help,$
        oldr=oldr,oldg=oldg,oldb=oldb, XYOUTS keywords

parameters      NONE

keywords
        white   [INPUT] if set, chooses colors appropriate for a
                white background, as is the case for a postscript
                device
                * mind you, setting this does NOT force the background
                  color to be white.  it is just that colors appropriate
                  for a white background are loaded in.
                * if not set, will get automatically set if the device
                  is postscript
        brewer  [INPUT] if set, uses single-hue 9-class color sequences
                from Color Brewer to fill in the 11-19, 21-29, 31-39,
                51-59, 81-89, slots
                (Cynthia Brewer et al., http://colorbrewer2.org)
        verbose [INPUT] controls chatter
        help    [INPUT] if set, prints out the calling sequence
                and notes which color is loaded into which index
                * automatically set if VERBOSE > 50
        oldr    [OUTPUT] old R colors
        oldg    [OUTPUT] old G colors
        oldb    [OUTPUT] old B colors
        _extra  [INPUT ONLY] pass defined keywords such as CHARTHICK,
                CHARSIZE, and ALIGN to XYOUTS
                * matters only if VERBOSE.ge.10, when a plot
                  showing all the colors and names is displayed.

description
        this is based on Deron Pease's scheme of assigning specific
        colors to color table indices and more to the point, being
        able to remember which color goes where.  Deron's idea is
        to have 1=BLUE, 2=RED, 3=GREEN and then have 10's to be
        different shades of blue, 20's to be shades of red, and 30's
        to be shades of greens, etc.  Further, within each dex, the
        colors run as: X0=base, X1=dark, X9=light.
```

```
        This basic scheme has been enhanced as follows:
        -- added 4=YELLOW, 5=PINK, 6=CYAN, 7=BROWN, 8=SAFFRON, 9=GREY
        -- the shades are optimised for different backgrounds
        -- if WHITE is set, then
           X1=light and X9=dark, and
           (4) <-> (7)
        To obtain the complete set of color shades used, set the
        keyword HELP in conjunction with VERBOSE>5

        color indices 100+ are not touched, but all else is
        overwritten.

example
        peasecolr,verbose=10
        peasecolr,/help,charthick=3
        peasecolr,verbose=10,/white,/help & setkolor,'white',0

history
        vinay kashyap (Jul02)
        returns to caller on display error (VK; Sep02)
        !D.N_COLORS is useless on VNC? (VK; May04)
        added saffron (VK; May05)
        added grey (VK; Aug12)
        added keyword BREWER (VK; Sep15)
```

B.174 PICKRANGE [/util]

```
function        pickrange
        interactively select intervals in data -- returns a long-integer
        array of position indices of selected bins

syntax
        oo=pickrange(x,y,sigmay=sigmay,xsize=xsize,ysize=ysize,wid=wid,$
        /help,dynrng=dynrng,markx=markx,marky=marky,markp=markp,markc=markc,$
        marks=marks,marko=marko,markl=markl,/legg,verbose=verbose,$
        xo=xo,yo=yo,oxr=oxr,oyr=oyr,xtitle=xtitle,ytitle=ytitle,$
        title=title,opx=opx,opy=opy,opc=opc,$
        /nuthin,/noday,/notime,/nouser,/nopack,stacol=stacol,$
        stasiz=stasiz,stathk=stathk)

parameters
        x       [INPUT; required] points where "reference curve" is defined
        y       [INPUT] the reference curve (usually data)
                * if not given, X is taken to be Y, and the indices of X are
                  taken to be X!

keywords
        sigmay  [INPUT] if given, plots error bars on Y
        xsize   [INPUT] width of window to display curve in
                * ignored if WID is not set
        ysize   [INPUT] height of window to display curve in
                * ignored if WID is not set
        wid     [INPUT] window ID
                * use current/default window if -ve/0
        help    [INPUT] prints out usage summary
        dynrng  [INPUT; default: 3 dex] maximum dynamic range in log scale
        markx   [INPUT] mark these x-coordinate points
        marky   [INPUT] mark at these y-coordinate positions
                * ignored if incompatible with MARKX
                * if not given, labels and marks will be placed 2/3 of
                  the way up the plot.
        markp   [INPUT: default: 1] PSYMs for each (MARKX,MARKY) pair
        markc   [INPUT: default: 222] COLOR for each (MARKX,MARKY) pair
        marks   [INPUT: default: 1] SYMSIZE for each (MARKX,MARKY) pair
        marko   [INPUT; default: 90 deg] ORIENTATION for MARKL
                * MARK[P,C,S,O]($) overrides the hardcoded defaults
        markl   [INPUT] label for each (MARKX,MARKY) pair
                * marked only if given, unless there's only one element,
                  in which case it is accepted as default
        legg    [INPUT] if set, drops a leg to the X-axis
```

```
        xo      [I/O] returns the x-position of last cursor click
        yo      [I/O] returns the y-position of last cursor click
        oxr     [I/O] 2-element array containing the final XRANGE
        oyr     [I/O] 2-element array containing the final YRANGE
        xtitle  [INPUT] passed to PLOT w/o checking
        ytitle  [INPUT] passed to PLOT w/o checking
        title   [INPUT] passed to PLOT w/o checking
        opx     [INPUT] X-coords of a curve to be overplotted
        opy     [INPUT] the curve to be overplotted
                * OPX and OPY may be 2-D, in which case the 1st dimension
                  defines the number of curves to be overplotted
                * sizes of OPX and OPY >must< match
        opc     [INPUT] color for the overplotting
                * default is 222
        verbose [INPUT] controls chatter
                * replaces QUIET

        _ref_extra      [INPUT ONLY] pass defined keywords to
                STAMPLE: NUTHIN,NODAY,NOTIME,NOUSER,NOPACK,STACOL,STASIZ,STATHK

restrictions
        * works only under X

usage summary
        see variable USAGE below

history
        vinay kashyap (Oct96)
        added keyword SIGMA; changed to return only position indices of
          selected bins; modified click/drag behavior (VK; Dec96)
        added keyword MARKX (VK; Feb97)
        added keywords MARKY, MARKP, MARKC, MARKS, MARKL, MARKO, LEGG, QUIET,
          XO, YO, OXR, OYR; modified behavior of LEFT CLICK; added keyboard
          control via LEFT CLICK (VK; Aug98)
        allowed XO,YO,OXR,OYR to be I/O (VK; Sep98)
        now MARKL(0) does not act as default, XYOUTS forced to CLIP (VK; Nov98)
        now zoom is remembered if OXR or OYR are set on input; made XR,YR
          double precision; MARKY bug corrected; renamed keyword SIGMA as
          SIGMAY; changed _EXTRA to _REF_EXTRA (VK; Dec98)
        bug correction: reversed X-axis was preventing zoom (VK; Apr99)
        added keywords OPX,OPY,OPC (VK; Oct99)
        cursor changes shape when mouse button is pressed (VK; FebMM)
        undid cursor shape changes (VK; AprMM)
        MARKC can now be I*4; added extra keyboard options; added call to
          STAMPLE (VK; DecMM)
```

```
changed MARKY default from 0.5 to 0.67 (VK; JanMMI)
added keyword TITLE; improved color-scale setting for 24-bit
  consoles (VK; FebMMI)
streamlined X,Y input behavior (VK; May02)
handle color tables in 24-bit displays (VK; Jun02)
replaced keyword QUIET with VERBOSE, changed color defaults
  (VK; Jul02)
speeded up marker plots if no variations exist (VK; Dec02)
button press status now stored in !MOUSE, not !ERR (VK; Apr09)
```

B.175 PLOTEM [/]

```
procedure       plotem
        plots emission measures derived from the output of POTEM_TOOL
        (or other means)

syntax
        plotem,logT,EM,idstr=idstr,markmin=markmin,multid=multid,lcol=lcol,$
        lsiz=lsiz,tlog=tlog,wdth=wdth,xtitle=xtitle,ytitle=ytitle,title=title,$
        xrange=xrange,yrange=yrange,xstyle=xstyle,ystyle=ystyle,/xlog,/ylog,$
        /nuthin,/noday,/notime,/nouser,/nopack,stacol=stacol,stasiz=stasiz,$
        stathk=stathk

parameters
        logT    [INPUT] temperatures at which to plot the emission measure
                estimates
        EM      [INPUT] array of emission measures at the various temperatures
                for each line
                * if EM is not given, it is assumed that EM <- "LOGT" and
                  LOGT = default
                * EM is an array of the form EM(LOGT,WVL); if the size of
                  the first dimension does not match input LOGT, the plots
                  are interpolated

keywords
        idstr   [INPUT] ID structure, essentially to figure out labels for
                the curves.
                * if size does not match that of EM, cest la vie.
        markmin [INPUT] if set, plots only a single point for each curve,
                corresponding to the minimum in EM
                * the actual value of MARKMIN sets the PSYM for the points
        multid  [INPUT] an integer array saying how many IDs there are
                to each component of the ID structure (comes in handy
                if IDSTR has all the emissivities collapsed into one)
        lcol    [INPUT; default=200] color for the labels
        lsiz    [INPUT; default=0.4] size of the labels
        tlog    [I/O] if LOGT does not match size of EM, >and< EM does
                not match the size of the temperature grid in emissivities
                in ID being considered, then look for TLOG to supply the
                actual temperature grid. (this latter feature is not yet
                implemented)
                * default is to go from log(T)=4..8 in as equidistant steps
                * beware -- this will get overwritten if incorrect!!
        wdth    [INPUT] the widths of each of the EM curves, from POTTASCH(),
                via POTEM_TOOL()
```

```
                * the size of this array must match the 2nd dimension of
                  EM, or else it will be ignored

        xtitle  [INPUT] passed w/o comment to PLOT
        ytitle  [INPUT] passed w/o comment to PLOT
        title   [INPUT] passed w/o comment to PLOT
        xrange  [INPUT] overrides locally determined XRANGE
        yrange  [INPUT] overrides locally determined YRANGE
        xstyle  [INPUT] passed w/o comment to PLOT
        ystyle  [INPUT] passed w/o comment to PLOT
        xlog    [INPUT] passed w/o comment to PLOT, but default is 0
        ylog    [INPUT] passed w/o comment to PLOT, but default is 1
        _extra  [INPUT] pass defined keywords to subroutines
                STAMPLE: NUTHIN,NODAY,NOTIME,NOUSER,NOPACK,STACOL,STASIZ,STATHK

history
        vinay kashyap (Nov98)
        added keyword MULTID (VK; Dec98)
        changed call to INITSTUFF to INICON (VK; 99May)
        added call to STAMPLE (VK; JanMMI)
        improved color-scale setting for 24-bit consoles (VK; FebMMI)
        added keyword WDTH (VK; SepMMVII)
```

B.176 PLOTFIT [/fitting]

```
procedure       plotfit
        takes the output of FITLINES and makes a nice plot of it
        with the data and the model(s) and everything.

syntax
        plotfit,x,y,fitstr,ymod,ysig=ysig,conlev=conlev,consig=consig,$
        funcs=funcs,intens=intens,scol=scol,ecol=ecol,ccol=ccol,mcol=mcol,$
        ncol=ncol,imcol=imcol,clin=clin,mlin=mlin,nlin=nlin,$
        missing=missing,fwhm=fwhm,betap=betap,angle=angle,submodel=submodel,$
        /nuthin,/noday,/notime,/nouser,/nopack,stacol=stacol,stasiz=stasiz,$
        stathk=stathk,$
        PLOT_KEYWORDS

parameters
        x       [INPUT; required] absissa, e.g., wavelength or energy
        y       [INPUT; required] count spectrum Y(X)
                * size of Y must match that of X
        fitstr  [INPUT] output of FITLINES
        ymod    [OUTPUT] model function

keywords
        ysig    [INPUT] errors on Y
                * default is 0.
                * plotted if supplied, otherwise not.
        funcs   [INPUT] name of user defined function that takes as input
                X and A (the model parameters), and returns Y(X;A) and
                the partial derivatives dY/dA
                * default is set to X3MODEL
        intens  [INPUT] if set, Y(X) is assumed to be intensity, i.e.,
                (counts|ergs|.../bin_unit/...)
                * default is to take Y(X) to be (counts|ergs|.../bin/...)
                * passed straight to FITLINES_EVENT
                * WARNING: this has >not< been road-tested!
        conlev  [INPUT] the continuum that was taken out of the spectrum
                * CONLEV must match the size of X and Y else ignored
                * if ignored, interpolated from FITSTR.CONLEV
        consig  [INPUT] error on CONLEV
                * default for CONSIG is sqrt(abs(CONLEV)+0.75)+1.
                * currently not implemented.  i.e., it's not plotted even
                  if given.
        scol    [INPUT; default=6] color of spectrum plot
        ecol    [INPUT; default=SCOL] color of error-bars
        ccol    [INPUT; default=1] color of continuum
```

```
        mcol    [INPUT; default=2] color of best-fit model+continuum
        ncol    [INPUT; default=3] color of best-fit model alone
        imcol   [INPUT; default=4] color to plot model components
                * individual model components are not plotted unless
                  IMCOL is defined and non-zero
        clin    [INPUT; default=1] line style of continuum
        mlin    [INPUT; default=0] line style of model+continuum
        nlin    [INPUT; default=2] line style of model alone

        missing [INPUT] passed w/o comment to FUNCS
        fwhm    [INPUT] passed w/o comment to FUNCS
        betap   [INPUT] passed w/o comment to FUNCS
        angle   [INPUT] passed w/o comment to FUNCS
        submodel[OUTPUT] the model function for each component,
                returned as a (NCOMP,NX) sized array
                * calculated iff IMCOL is set

        _extra  [INPUT ONLY] use this to pass defined keywords to
                PLOT and STAMPLE

restrictions
        requires subroutines
                FUNCS (X3MODEL [MK_3MODEL, [MK_GAUSS, MK_LORENTZ]])
        PLOT cannot handle any keyword that is either strange or undefined

known bugs
        cannot handle spectra with varying bin sizes
        INTENS has not been tested

history
        vinay kashyap (MarMM)
        added call to STAMPLE (VK; FebMMI)
        bug correction; changed default colors; added keyword IMCOL
          (VK; May'03)
        added keyword SUBMODEL (VK; NovMMVI)
```

B.177 POAINTSYM [/util]

```
procedure       poaintsym
        Define useful plotting symbols not in the standard !PSYM
        definitions.  After symbol has been defined with this
        routine, a plotting command should follow with either
        PSYM = 8 or !P.PSYM = 8 (see USERSYM).

syntax
        poaintsym,symb,psize=psize,pthick=pthick,/pfill,color=color,$
        pie=pie,/halftri,npoint=npoint,aspect=aspect,angle=angle,adent=adent,$
        verbose=verbose

parameters
        symb    [INPUT; required] string describing the type of
                symbol to define.  The following are accepted:
                  ’circle’
                  ’dash’
                  ’star’ ’starfish’ ’asterisk’
                  ’mercury’ ’venus’ ’earth’ ’moon’ ’mars’
                  ’jupiter’ ’saturn’ ’uranus’ ’neptune’ ’pluto’
                  ’square’ ’diamond’ ’box’ ’spoke’
                  ’triangle’ ’down triangle’ ’right triangle’ ’left triangle’
                  ’down arrow’ ’up arrow’ ’left arrow’ ’right arrow’
                  ’upper limit’ ’lower limit’
                  ’tilde ANGLE OVERFLOW’
                * if something unknown is specified, the program
                  prints a list of accepted options and quits
                  without making a change
                * special cases:
                  -- STARFISH: is 5-pointed
                  -- ASTERISK: is (NPOINT/2)-pointed and uses keyword
                     ADENT to figure out indentation
                  -- EARTH: forces PFILL=0
                  -- MOON: use ADENT to pull the horns out, and PHASE to
                     control thickness
                  -- URANUS: forces PFILL=0
                  -- PLUTO: forces PFILL=0
                  -- SPOKE: essentially DIAMOND with an aspect x:y::1:10
                  -- UPPER/LOWER LIMIT: synonyms for DOWN/UP ARROW respec.
                  -- TILDE: set ANGLE (default=0) to rotate the symbol,
                     and OVERFLOW (default=1) to pull the ends out a bit

keywords
        psize   [INPUT] size of the plotting symbol in
```

```
                multiples of the standard size (see SYMSIZE); does
                not need to be an integer
        pthick  [INPUT] thickness of symbol being drawn
        pfill   [INPUT] if set, fills the symbol
                * does not affect arrows or character symbols
        pie     [INPUT] range of angles in [degrees] to include while
                making a circle
                * if scalar, includes from PIE[0] to 360.
                * if vector, includes from PIE[0] to PIE[1]
        halftri [INPUT] if set, lops off the triangle such that only the
                part that protrudes past the axes (above x-axis for up
                triangle, below x-axis for down triangle, etc.) is plotted
        npoint  [INPUT] number of vertices in the symbol
                * note that usersym maximum is 49
                * currently, only matters to TILDE
        aspect  [INPUT] changes the aspect ratio of the symbols
                by scaling the axes prior to rotation
                * if +ve, scales X-axis
                * if -ve, scales Y-axis
                * if array, only the first element is used
                * if 0, nothing is done
        rotate  [INPUT] angle by which to rotate the symbol [deg]
        adent   [INPUT] the magnitude of the indentation for STARFISH
                and ASTERISK and the extent of the horns for MOON
                * default is 0.4 for STARFISH and ASTERISK,
                  and 10 for MOON
        phase   [INPUT] controls thickness of moon
                * default is 0
        verbose [INPUT] controls chatter
        _extra  [INPUT ONLY] pass defined keywords to subroutines
                USERSYM: COLOR

method
        Appropiate X,Y vectors are used to define the symbol and passed
        to USERSYM

examples
        Plot with filled stars as the symbol, twice the default size
          POAINTSYM,'star',psize=2,/pfill & PLOT,findgen(10),PSYM=8
        Plot with open circle as the symbol
          POAINTSYM,'circle' & PLOT,findgen(10),PSYM=8
        Plot filled color circles
          POAINTSYM,'circle',/pfil,psize=3,col=2 & PLOT,findgen(10),PSYM=8
        Plot arrows all around
          POAINTSYM,'left arrow',pthick=2,psize=2 & plot,findgen(11),psym=8
```

```
          POAINTSYM,'up arrow',pthick=3,psize=3 & oplot,findgen(11),psym=8
          POAINTSYM,'right arrow',pthick=4,psize=4 & oplot,findgen(11),psym=8
          POAINTSYM,'down arrow',pthick=5,psize=5 & oplot,findgen(11),psym=8
        Plot tildes
          POAINTSYM,'tilde',pthick=5,psize=5 & plot,findgen(11),psym=8
          POAINTSYM,'tilda 90',pthick=5,psize=5 & plot,findgen(11),psym=8
          POAINTSYM,'tilda 90 3',pthick=5,psize=5 & plot,findgen(11),psym=8
          POAINTSYM,'tilda 0 20',pthick=2,psize=5,npoint=49 & plot,findgen(11),psym=8

history
        modified from JDSYM (Jeremy Drake 2001; JJD added square and
          diamond symbols to PLOTSYM [ASTROLib; Wayne Landsman 1992])
          by Vinay Kashyap (Oct03)
        added option TILDE and keyword NPOINT (VK; Feb04)
        added keywords ASPECT, ROTATE, and ADENT; added options SPOKE,
          MERCURY, VENUS, EARTH, MOON, MARS, JUPITER, SATURN, URANUS,
          NEPTUNE, PLUTO (VK; Jun05)
        added keyword PIE (VK; Aug12)
        added option DASH (VK; Jul15)
```

B.178 POPLEV [/mkemis]

```
function        poplev
                returns the level populations at given temperature and
                pressure for a set of ions of a given element

usage
        pop=poplev(temp,pres,chianti,n_e=n_e)

parameters
        temp    [INPUT; required] Temperature [K]
        pres    [INPUT; required] pressure [cm^-3 K]
        chianti [INPUT; required] structure that contains all the relevant
                information gleaned from the CHIANTI database (type
                        DBHELP=RD_CHIANTI() & HELP,DBHELP,/STRUCTURE
                for a description of what this structure contains)

keywords
        n_e     [INPUT] electron density in cm^-3
                * if set, overrides PRES/TEMP

requires
        DESCALE2_UPS, a CHIANTI routine

history
        originally written as POPULATE.PRO, a CHIANTI routine,
                by Ken Dere (v2: Mar96)
        modified to eliminate common blocks, mesh with SCAR, and
        fltarr->dblarr (to remove the -ve population levels)
                by Vinay Kashyap (Nov96)
```

B.179 POPSOL [/mkemis]

```
function        popsol
        returns the level populations at given temperature and
        pressure for a set of ions of a given element as an array
        of size [N(TEMP),N(PRES or DENSITY),N(N_LEVELS)]

usage
        pop=popsol(temp,pres,chianti,n_e=n_e,/delec,$
        radtemp=radtemp,dilute=dilute,pe_ratio=pe_ratio,$
        n_levels=n_levels,data_str=data_str)

parameters
        temp    [INPUT; required] Temperature [K]
        pres    [INPUT; required] pressure [cm^-3 K]
        chianti [INPUT; required] structure that contains all the relevant
                information gleaned from the CHIANTI database (type
                        DBHELP=RD_CHIANTI() & HELP,DBHELP,/STRUCTURE
                for a description of what this structure contains)

keywords
        n_e     [INPUT] electron density in cm^-3
                * if set, overrides PRES/TEMP
        delec   [INPUT] if set, assumes these are DR lines
        z       [INPUT] atomic number of element
        jon     [INPUT] ionic state of species
                * Z and JON are used only if DELEC is set
        radtemp [INPUT] The blackbody radiation field temperature
                * default is 6000 K
        dilute  [INPUT] a dilution factor for photoionization, based
                on the distance above a star's surface -- see CHIANTI
                routine R2W()
                * if not set, assumed to be 0
        pe_ratio [INPUT] proton-to-electron ratio
                * if not given, assumed to be 1.0
        n_levels [INPUT] number of levels in the model
                * (from CHIANTI/idl/low_level/pop_solver.pro)
                This allows the number of levels in the model to be reduced.
                E.g., if the full model contains 100 levels, one could set
                N_LEVELS=50. This can be useful if one is interested in
                looking at the effects of cascading from higher levels.
        data_str [OUTPUT] some potentially useful data
                * (from CHIANTI/idl/low_level/pop_solver.pro)
                If POP_SOLVER is called for just 1 temperature and density,
                then the individual data arrays for each of the physical
```

```
                processes can be output through DATA_STR.  This allows the
                user to check for the dominant processes affecting the
                population of a given level. DATA_STR is a structure with
                the following tags:
                        .aa     A-values
                        .aax    Photoexcitation/stimulated emission
                        .cc     Electron rate coefficients
                        .ccp    Proton rate coefficients
                Each tag is a 2D array, and is such that, e.g.,
                aa[0,20] corresponds to an excitation, while aa[20,0]
                is a de-excitation.

A note about the default value of PE_RATIO:
        The original version of this program,
                CHIANTI/dbase/idl/low_level/pop_solver.pro,
        set the default to be 0.83, which is the limit for a high-T
        plasma with cosmic abundances.  Later versions include a
        calculated value passed in via a common block.  Here we
        exclude this factor because we want to store the emissivities
        without reference to their ion balance and abundance, on which
        this factor depends.  The intensity of a line transition from
        upper level j to lower level i is in general
                dI(lam_ij) = (1/4pi)(hc/lam_ij) N_j A_ji dV [ergs/s/sr]
        where lam_ij is the wavelength of the transition, A_ji is the
        Einstein coefficient (the spontaneous transition probability),
        dV is the emitting volume, and N_j is the number density of the
        ions in level j.  We can write N_j as the product
                N_j = (N_j/N_XIon) (N_XIon/N_X) (N_X/N_H) (N_H/N_e) N_e
        where N_XIon is the number density of the ions,
        N_j/N_XIon is the population fraction of the ions at level j,
        N_XIon/N_X is the ion fraction,
        N_X/N_H is the relative abundance of element X wrt H, and
        N_H/N_e is the H density relative to the free-electron density.

        In PINTofALE, we calculate and store the emissivities as
                e_ji = (hc/lam_ij) A_ji (1/N_e) (N_j/N_XIon)
        and write the line intensity as
                f(lam_ij) = e_ji (N_XIon/N_X) (N_X/N_H) EM (N_H/N_e)
        where EM = N_e^2 V is the emission measure, and the factor
        (1/4pi) is always ignored and the factor (hc/lam_ij) is usually
        taken out again.  This calculation is carried out in LINEFLX(),
        which provides for the inclusion of the ion balance, abundance,
        and the proton-to-electron ratio.  Hence the use of a default of
        1.0 for PE_RATIO here.
```

```
subroutines
        CHIANTI routine DESCALE_ALL
        NENH

history
        POP_SOLVER.PRO written by Peter Young
{..
..      Ver 1, PRY 29-Mar-99
..      Ver 2, PRY 30-Apr-99, added call to get_prot_rates
..      Ver 3, PRY 15-Dec-99, added deu to upsilon common block in order
..              to be consistent with the main Chianti routines.
..      Ver 4, PRY 9-May-00, corrected problem with threshold when dealing
..              with sparse matrices. Basically values less than 1.e-30 in
..              the c-matrix were being set to zero and giving rise to
..              NaN's in certain circumstances.
..       Ver.5, PRY 14-Jul-00, changed elvl common block to the elvlc common
..               block which is now the Chianti standard. Also, when
..               descaling upsilons, the routine now uses the Delta-E from
..               the .splups file.
..       Ver.6, PRY 9-Aug-00, changed routine to deal better with the
..               dielectronic recombination files
..       Ver.7, PRY 17-Aug-00, routine does not call LINBCG now if radtemp
..               is non-zero.
..       Ver.8, PRY 29-Aug-00, the sparse matrix section has been disabled.
..       Ver.9, PRY 12-Nov-01, calls routine proton_dens() to calculate the
..               proton to electron ratio.
..       Ver.10, PRY, 6-Dec-01, corrected bug when there are more levels
..               in .splups file than in .elvlc file (ZnXXV).
..       Ver.11, PRY, 11-Jul-02, removed ION keyword
..       Ver.12, PRY, 9-Aug-02, within the equation solving section, I've set
..               the population of the ground level (rather the n_level level)
..               to 1, and this seems to stop negative populations appearing
..               in extreme conditions.
..       Ver.12, PRY, 21-Aug-02, changed exp(-1/1/a) to exp(-a) in electron
..               excitation section which caused a hang-up in some
..               circumstances. Also, the routine now uses vector ECMC
..               (combined experimental and theoretical energies) in
..               determining if a level lies above or below another level.
..               Previously only used the observed energy vector. Also, the
..               exponential in the electron excitation section now uses the
..               (accurate) .elvlc energy separation rather than the .splups
..               energy separation, which can cause significant (~20-30%)
..               differences in level populations of high-lying levels at
..               low temperatures.
..       Ver.13, PRY, 10-Sep-02, corrected bug for proton rates. The excitation
```

```
..               and de-excitation rates were being swapped.
..
..       V. 14  4-Oct-2003  Giulio Del Zanna (GDZ).
..               -removed all COMMON blocks (note that only proton_dens.pro has
..                one: COMMON elements,abund,abund_ref,ioneq,ioneq_logt,ioneq_ref)
..               -only the essential information input is passed to the routine
..                via a new input structure.
..               -fixed a bug, that affected all the satellite lines, and was
..                introduced in v.12,  included in  CHIANTI v.4.0.
..                basically the ionization potential was not subtracted when
..                calculating the Delta E in the exponential.
..
.. VERSION     : 14, 4-Oct-2003
..}
        modified to eliminate common blocks and the call to SETUP_ION,
          and altered the I/O interface to mesh with PINTofALE, changed
          call to PROTON_DENS() to call NENH() and changed default PE_RATIO
          to 1.0 (Vinay Kashyap, Apr03)
        modified to account for v14, for CHIANTI v4.2 (VK; Apr04)
```

B.180 POTEM_TOOL [/]

```
function        potem_tool
        return emission measures EM(LOGT,WVL) for all lines that have been ID'd
        (but see keyword OKMULT)

        for each temperature, assume a delta-function emission measure,
        compute line fluxes seen through some instrument, account for
        interstellar absorption, and scale to match the observed flux.

        in case of multiple IDs for a single observed line, the emissivities
        of the IDs are added together, and assumed for all practical purposes
        to be the strongest line modified by the other IDs.
        in case of IDs spanning different elements, the emissivities are
        added as before, but weighted with their relative abundances.

syntax
        EM=potem_tool(idstr,flux,logT=logT,abund=abund,ldir=ldir,$
        dwvl=dwvl,/okmult,/kettle,outid=outid,wdth=wdth, pres=pres,$
        logP=logP,n_e=n_e, chifil=chifil,chidir=chidir,eqfile=eqfile,$
        NH=NH,/noph,defEM=defEM, thresh=thresh,/temp,/ikev,effar=effar,$
        wvlar=wvlar,fh2=fh2,He1=He1,HeII=HeII,/fano, level=level,$
        verbose=verbose)

parameters
        idstr   [INPUT; required] the ID structure that comes out of LINEID
        flux    [INPUT] observed fluxes
                * if not given, taken from IDSTR.(*).FLUX
                  (if zero, forcibly set to 1 [(ph|erg)/s/cm^2])
                * size is forced to match IDSTR.WVL
                  if <, filled out with 1s
                  if >, extra elements are ignored

keywords
        logT    [INPUT] temperatures at which to compute the emission measures
                * default is findgen(81)*0.05+4.
        abund   [INPUT] abundances
                * default is to use Anders & Grevesse
        ldir    [INPUT] string array containing line database directory
                names, one for each ID.
                * if given, will NOT use the embedded emissivities in
                  IDSTR.EMIS (default is to use them)
                * if size < N(ALLMATCHES), the last element is replicated
                  as often as necessary
                * to selectively use IDSTR.(#).EMIS, set to '0' within a
```

```
                 string array (e.g., ['$CHIANTI','$SPEX','0','0','$SCAR']
               * if set to a number, assumes "$SCAR"
        dwvl    [INPUT; default=5e-3] search the line database for the
                wavelength of interest with this assumed roundoff error
        okmult  [INPUT] if set, then multiple IDs (if any) are >not<
                collapsed, and the output will be of the form EM(LOGT,ALLWVL)
                * automatically set if size of FLUX matches the total number
                  of matches
        kettle  [INPUT] if set, does NOT do Pottasch corrections
        outid   [OUTPUT] if defined on input, returns the IDs with new
                emissivities in an ID structure
                * multiple IDs are "collapsed" unless OKMULT is set
        multid  [OUTPUT] number of IDs for each observed line (this is
                of use in, e.g., PLOTEM)
        wdth    [OUTPUT] the full width in logT of the emissivity curve,
                as calculated by POTTASCH()
        verbose [INPUT] controls chatter
        _extra  [INPUT ONLY] use to pass defined keywords to subroutines:
                RD_LINE: PRES,LOGP,N_E,VERBOSE
                FOLD_IONEQ: CHIFIL,CHIDIR,EQFILE,VERBOSE
                FLUX_TO_EM: NH,NOPH,DEFEM,THRESH
                LINEFLX: TEMP,IKEV,EFFAR,WVLAR
                ISMTAU: FH2,HE1,HEII,FANO
                POTTASCH: LEVEL

subroutines
        INICON
        RD_LIST
            RD_LINE
            FOLD_IONEQ
                READ_IONEQ
                    RD_IONEQ (CHIANTI routine)
        FLUX_TO_EM
            LINEFLX
            ISMTAU
        POTTASCH

history
        vinay kashyap (Nov98; "concatenated" id_to_emis and flux_to_em)
        changed default behavior of FLUX; added keyword MULTID; changed
          OUTID.().Z and OUTID.().ION to floats in case multiple IDs have
          been collapsed to one emissivity (VK; Dec98)
        added field FLUXERR to output IDSTR (VK; Mar99)
        changed call to INITSTUFF to INICON (VK; 99May)
        no need to read in ion balance info in POTTASCH again (VK; JanMMI)
```

```
added keyword WDTH to carry the emissivity width info outside
  (VK; SepMMVII)
```

B.181 POTTASCH [/]

```
function        pottasch
        returns the Pottasch factor that is a measure of the width of
        the ion emissivity in temperature space over which most of
        the line emission occurs.

        the Pottasch factor is the ratio
                {\int [EMIS*ION_BALANCE] dlogT}/{WDTH*max([EMIS*ION_BALANCE])}
        where WDTH is the range in logT where EMIS*ION_BALANCE drops to a
        set fraction of the maximum.

        output has as many elements as the 2nd dimension of EMIS, one point
        for each line.

syntax
        pot=pottasch(emis,logt,z=z,ion=ion,level=level,wdth=wdth,$
        trange=trange,chifil=chifil)

parameters
        emis    [INPUT; required] 1- or 2-D array of line intensities as
                a function of temperature
                * if 1D, assumed to be EMIS(Temperature)
                * if 2D, assumed to be EMIS(Temperature,{Z,ION,Wavelength})
                * if Z is not specified, assumed to be the product of
                  line emissivity and ion balance
        logt    [INPUT] 1D array of log10(T[K])
                * WARNING: if not given, or if size does not match 1st dim
                  of EMIS, makes a dumb guess

keywords
        Z       [INPUT] atomic numbers corresponding to the 2nd dimension
                * if not set, ION is ignored and FOLD_IONEQ is not called
                * if size does not match 2nd dim of EMIS,
                  >: ignore extra elements
                  <,>1: use as many as specified, take rest to be H (Z=1)
                  1: all are of same element
        ion     [INPUT; default: Z+1] ionic state
        level   [INPUT; default: 0.1] defines what WDTH means
        wdth    [OUTPUT] full width in logT at LEVEL*maximum
                * minimum value at any point is the resolution in LOGT at
                  that point.
        trange  [OUTPUT] 2-column array of the boundaries of logT that
                define WDTH
                * trange(0,*)=lower bound, trange(1,*)=upper bound
```

```
        _extra  [INPUT] use to pass defined keywords
                * currently, only CHIFIL to FOLD_IONEQ

subroutines
        FOLD_IONEQ [WHEE, RD_IONEQ [READ_IONEQ]]

history
        vinay kashyap (Apr97)
        changed temperature width calculation to account for bumpy
          emissivities (VK; Oct98)
```

B.182 PPD_SRC [/stat]

```
function        ppd_src
        returns the posterior probability density for the source
        intensity, marginalized over background intensity, assuming
        gamma-function priors (see van Dyk et al., 2001, ApJ 548, 224)

        D:ns,nb ; I:asrc,abkg,f ; M:s,b ; p(M|DI)=p(M|I)*p(D|MI)/p(D|I)

        r = abkg/asrc & A = nb+alfa_B & B = f*(beta_B+r)

        p(b|I) = ((f*beta_B)^alfa_B/Gamma(alfa_B)) * b^(alfa_B-1) * exp(-b*beta_B*f)
        p(nb|b,I) = ((r*f*b)^nb/Gamma(nb+1)) * exp(-b*f*r)
        p(b|nb,I) = (B^A/Gamma(A)) * b^(A-1) * exp(-b*B)

        I_k = b^(ns+A-k-1) * s^(k+alfa_S-1) * exp(-s*f*(1+beta_S) -b*(f+B)) /
                (Gamma(k+1)*Gamma(ns-k+1))
        I_kb = (Gamma(ns+A-k)/Gamma(k+1)/Gamma(ns-k+1)) * (f+B)^(-(ns+A-k)) *
                s^(k+alfa_S-1) * exp(-s*f*(1+beta_S))
        I_kbs = (Gamma(ns+A-k)*Gamma(k+alfa_S)/Gamma(k+1)/Gamma(ns-k+1)) *
                (f+B)^(-(ns+A-k)) * (f*(1+beta_S))^(k+alfa_S)

        p(s|I) = ((f*beta_S)^alfa_S/Gamma(alfa_S)) * s^(alfa_S-1) * exp(-s*beta_S*f)
        p(ns|M,I) = (f*(s+b))^ns/Gamma(ns+1) * exp(-(s+b)*f)
        p(sb|D,I) = p(b|nb,I)*p(s|I)*p(ns|M,I) / p(ns|I)
                  = (Sum_{k=0}^{ns} I_k) / (Sum_{k=0}^{ns} I_kbs)

        p(ns|I) = f^ns * (B^A/Gamma(A)) * ((f*beta_S)^alfa_S/Gamma(alfa_S)) *
                Sum_{k=0}^{ns} I_kbs
        p(s|D,I) = (Sum_{k=0}^{ns} I_kb) / (Sum_{k=0}^{ns} I_kbs)

syntax
        ppd=ppd_src(ns,nb,sgrid,smap=smap,scred=scred,clev=clev,ppdstr=ppdstr,$
        funit=funit,asrc=asrc,abkg=abkg,agamma=agamma,bgamma=bgamma,$
        priorbg=priorbg,nsgrid=nsgrid,srcmin=srcmin,srcmax=srcmax,verbose=verbose)

parameters
        ns      [INPUT; required] observed counts in source region
        nb      [INPUT; required] observed counts in background region
                * NS and NB must be scalar -- if vector, adds up all the elements
        sgrid   [OUTPUT] the output grid over which the PPD is calculated
                * a regular grid that uses NSGRID, SRCMIN, and SRCMAX

keywords
        smap    [OUTPUT] the mode of the distribution, corresponding to the
```

```
        maximum a posteriori value of the PPD
scred   [OUTPUT] a two-tailed credible range on the source intensity,
        defined to include CLEV/2 of the area under the PPD above
        and below SMAP
        * NOTE: what this means is, if the distribution is skewed,
          then the total area under PPD as defined by SCRED is not
          necessarily equal to CLEV
        * The one-tailed credible regions, that is, the range that
          encloses an area CLEV under the PPD, are returned in the
          structure PPDSTR.  these regions are determined such that
          the mode is always _included_ within the region, but is not
          necessarily at the center
clev    [INPUT] a nominal level at which to determine bounds and
        get credible regions
        * default is 0.68
        * if < 0, abs(CLEV) is used
        * if > 1 and < 100, then assumed to be given as a percentage
        * if > 100, then 1-1/CLEV is used
ppdstr  [OUTPUT] an anonymous structure containing the fields
        {
          GRID : same as SGRID
          MODE : mode, same as SMAP
          MEAN : mean of the distribution
          SIG : s.d. on the MEAN
          MEDIAN : the median of the distribution
          CLEV : the probability level defined by the keyword CLEV
          HPD1 : the highest posterior density "1-sigma" credible range,
                 the interval that encloses 68% of the area and also
                 the highest values of the probability density
          HPD2 : as HPD1, for "2-sigma", 95% of the area
          HPD3 : as HPD1, for "3-sigma", 99.7% of the area
          HPD0 : as HPD1, but corresponding to a fraction CLEV
          CRED1 : the "1-sigma" credible range, corresponding to 68%
                  of the area under the PPD that _includes_ the MODE
          CRED2 : as CRED1, for "2-sigma", 95% of the area
          CRED3 : as CRED1, for "3-sigma", 99.7% of the area
          CRED0 : as CRED1, but corresponding to a fraction CLEV of the
                  area under the PPD that includes the MODE
          RNG1 : the two-tailed "1-sigma" range, corresponding to a
                 fractional area under the PPD of 0.34 above and below
                 the MODE
          RNG2 : as for RNG1, but the "2-sigma" range, 0.475 of the area
          RNG3 : as for RNG1, but the "3-sigma" range, 0.4985 of the area
          RNG0 : as for RNG1, but CLEV/2 of the area, same as SCRED
          lnpD : the log of the predictive probability, ALOG(p(ns|I))
```

```
                }
        funit   [INPUT] a factor to _divide_ the counts to convert to different units
                e.g., ph/cm^2/s
                * assumed to be the same for both source and background counts
                  (any differences in exposure times can be encoded in ASRC and ABKG)
                * default is 1.0
        asrc    [INPUT] area (or exposure time) for the background contaminated
                source observation, NS
                * default is 1.0
        abkg    [INPUT] area (or exposure time) for the background observation, NB
                * default is 1.0
        agamma  [INPUT] alpha parameter for the source intensity prior, assumed to
                be a gamma distribution (see PROB_GAMMADIST)
                * default is 1.0 (also the hardcoded minimum)
                * generally taken to be the number of *expected* source counts+1
        bgamma  [INPUT] beta parameter for the source intensity prior
                * default is 0.0 (also the hardcoded minimum)
                * generally the ratio of that area in which the AGAMMA-1 counts are
                  obtained in, to ASRC
                * note that the default values of AGAMMA and BGAMMA make for a
                  very non-informative, but improper, prior
        priorbg [INPUT] the expected number of counts due to the background,
                a number that is known from prior information and NOT derived
                from NB itself.
                * a Gamma function prior is assumed on the background strength, with
                  alpha=priorbg+1; beta=abkg/asrc, but if priorbg=0, then beta=0
        nsgrid  [INPUT] number of points in output grid
                * default=101L
        srcmin  [INPUT] minimum in source intensity value to consider
        srcmax  [INPUT] maximum in source intensity value to consider
                * beware that setting [SRCMIN,SRCMAX] to too small a range will cause
                  it to act as a prior over and above the gamma prior.  this may not be
                  desirable.
                * the default range is the widest of:
                  -- NS +- 10*sqrt(NS)
                  -- (NS-(ASRC/ABKG)*NB) +- 10*sqrt(NS+(ASRC/ABKG)^2*NB) (if ABKG > 0)
                  -- (agamma/bgamma) +- 10*sqrt(agamma/bgamma^2) (if BGAMMA > 0)
                  as long as SRCMIN isn't -ve or SRCMAX is smaller than 20
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

subroutines
        PROB_GAMMADIST

history
```

```
vinay kashyap (May02)
added keyword FUNIT (VK; Oct03)
added HPD1,HPD2,HPD3,HPD0 to the output (VK; Mar06)
```

B.183 PRED_FLX [/]

```
function        pred_flx
        return line fluxes for a given DEM and compare with observed fluxes.

syntax
        f=pred_flx(line,logT,wvl,DEM,Z=Z,NH=NH,fobs=fobs,fsigma=fsigma,$
        ulim=ulim,verbose=verbose,effar=effar,wvlar=wvlar,fH2=fH2,He1=He1,$
        HeII=HeII,/fano)

parameters
        line    [INPUT; required] line cooling emissivities (T,Wvl) in units
                of 1e-23 erg cm^3/s, including ion balance, but not abundances.
        logT    [INPUT; required] array of log10(Temperature [K]) at which
                emissivities are given.
        wvl     [INPUT; required] wavelength of matching spectral line(s) [Ang]
        DEM     [INPUT] Differential Emission Measure at each T [cm^-5/logK]
                * passed w/o change to LINEFLX

keywords
        Z       [INPUT] atomic number of element that generates each WVL
                * passed w/o change to LINEFLX
        NH      [INPUT] H column density [cm^-2]
        fobs    [INPUT] observed fluxes; if supplied, displays comparison
                plots of predicted v/s observed
        fsigma  [INPUT] errors on observed fluxes; if not supplied, assumed
                to be 1+sqrt(abs(FOBS)+0.75)
        ulim    [INPUT] flag to indicate whether a given FOBS is an upper
                limit (1: UL, 0: not)
        verbose [INPUT] controls chatter
        _extra  [INPUT] allows setting defined keywords to
                LINEFLX [EFFAR, WVLAR, TEMP, ABUND, KEV, NOPH]
                ISMTAU [FH2,HE1,HEII,FANO]

restrictions
        * requires subroutines
          GETABUND
          LINEFLX [WHEE]
          ISMTAU

history
        vinay kashyap (May97)
        changed keyword name SIGMA to FSIGMA (VK; MMaug)
        changed Fobs/Fpred from /ylog to plotting alog10() (VK; MMVsep)
        changed Observed v/s predicted to plot psym=4
```

```
bug correction: matching up upper limit points (VK; MMVIapr)
added keyword VERBOSE (VK; MMVIIjun)
```

B.184 PROB_GAMMADIST [/stat]

```
function        prob_gammadist
        compute the gamma distribution, which is a continuous distribution
        on the real line with density function
                f(x) = b^a x^(a-1) exp(-bx) / Gamma(a)
        (see eqn 4 of van Dyk et al., 2001, ApJ 548, 224)
        this distribution has mean (a/b) and variance (a/b^2) for a,b>0

        This is equivalent to a Poisson distribution, and is often
        used as a prior function, with a and b representing our prior
        expectation that a-1 counts should be observed in an area
        b*src_area; a non-informative prior therefore has a=1 and b=0,
        corresponding to 0 counts in 0 area.

syntax
        g=prob_gammadist(x,agamma=agamma,bgamma=bgamma,verbose=verbose)

parameters
        x       [INPUT; required] array of abscissa values at which to
                compute the Gamma distribution
                * for x < 0, the function will return NaNs

keywords
        agamma  [INPUT; default=1] "alpha" parameter for the function
                * AGAMMA must be > 0, or else the function will return Infs
        bgamma  [INPUT; default=0] "beta" parameter for the function
                * BGAMMA should be .GE. 0, or else who knows what will happen
        verbose [INPUT; default=0] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (Aug01)
```

B.185 PROPERMOTION_CORRECTOR [/util]

```
procedure       propermotion_corrector
        computes proper motion and new coordinates

syntax
        propermotion_corrector.catra,catdec,pmra,pmdec,newra,newdec,$
        catyr=catyr,newyr=newyr,verbose=verbose,$
        /i24,/osxg,/hrs,/idnum,/idpre,/idform,/pm,/chop,/osep

parameters (all REQUIRED)
        catra   [INPUT] catalog RA
        catdec  [INPUT] catalog Dec
                * CATRA and CATDEC must have the same sizes
                * may be either float or string array
                * if float, assumed to be decimal degrees unless /I24 is set
                * if string, assumed to be decimal degrees unless ':' or ' '
                  are found
        pmra    [INPUT] proper motion in RA [mas/yr]
        pmdec   [INPUT] proper motion in Dec [mas/yr]
                * PMRA,PMDEC must be scalars,
                  or have the same size as CATRA,CATDEC
        newra   [OUTPUT] new RA
        newdec  [OUTPUT] new Dec
                * newRA, newDEC will have the same size as CATRA,CATDEC
                * default is to return decimal degrees
                * set keyword /OSXG to return the output in sexagesimal

keywords
        catyr   [INPUT] epoch of CATRA,CATDEC
                * default is 2000.0
        newyr   [INPUT] epoch for which to compute new coordinates
                * default is today
        verbose [INPUT] controls chatter
        i24     [INPUT] caught and passed on to SKYPOS's first run
                * use iff input are _not_ in decimal degrees
        osxg    [INPUT] caught and passed on to SKYPOS's second run
                * use iff you want the output in sexagesimal
        _extra  [INPUT ONLY] pass defined keywords to subroutines
                SKYPOS: HRS, IDNUM, IDPRE, IDFORM, PM, CHOP, OSEP

requires
        SKYPOS [DEG2HRS [STR_2_ARR, SYZE]]
```

```
history
        vinay kashyap (2014sep05)
```

B.186 RADPROJECT [/util]

```
function        radproject
        Projects a ray originating from an arbitrary point on an arbitrarily
        oriented sphere onto plane at \infty.  Returns a structure containing
        the radial coordinate, and the corresponding x-coords, y-,z-offsets,
        the magnitude of the projection, the corresponding volume of the
        pillbox, as well as all the input values.

syntax
        rstr=radproject(lat,lon,inrad=inrad,maxrad=maxrad,rrng=rrng,rdel=rdel,$
        opaq=opaq,arcone=arcone,theta0=theta0,phi0=phi0,$
        xout=xout,yout=yout,zout=zout,rproj=rproj,rout=rout,volout=volout,$
        verbose=verbose)

parameters
        lat     [INPUT; required] latitude on sphere at which ray is anchored
        lon     [INPUT; required] longitude on sphere at which ray is anchored
                * LAT,LON must be scalar
                * (LAT,LON)=(0,0) projects to (Y,Z)=(0,0)

keywords
        inrad   [INPUT; default=1] minimum r value of ray to consider
        maxrad  [INPUT; default=10] maximum r value of ray to consider
        rrng    [INPUT; default=[INRAD,MAXRAD]] filter the output by limiting
                the radii to consider
                * minmax(RRNG) is forced to not spill beyond [INRAD,MAXRAD]
                * use this only if you want everything outside this range to be
                  explicitly thrown away. discarded.  not zeroed, but removed
                  from the output.
        rdel    [INPUT; default=0.1] steps in r
        opaq    [INPUT; default=0] decides how to treat those portions of rays
                that are occluded by the sphere defined by radius INRAD
                - by default, this is assumed to be completely transparent
                - set OPAQ to negative number to set the opacity as exp(-OPAQ)
                - any value OPAQ>0 results in a completely opaque inner sphere
                  (so set /OPAQ to get an opaque inner sphere)
        arcone  [INPUT; default=1] area of the opening cone at radius=1 [deg^2]
        theta0  [INPUT; default=0] tilt of sphere [deg]
        phi0    [INPUT; default=0] twist of sphere [deg]
                * theta0 and phi0 are defined in a coordinate frame where the
                  projection plane is at x=\infty.
                * it is assumed that to get to the inclined sphere from the
                  observer's frame, you first apply THETA0 and then PHI0.
        rout    [OUTPUT] radii at which projections are computed
```

```
        xout    [OUTPUT] computed X
        yout    [OUTPUT] projected Y
        zout    [OUTPUT] projected Z
        rproj   [OUTPUT] projection of the radial unit vector towards the Y-Z
                plane, weighted by the opacity, at each ROUT
        volout  [OUTPUT] volume of the pillbox at each ROUT
        verbose [INPUT; default=0] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

example
        .run radproject

history
        vinay kashyap (2012dec; supersedes pathrofile)
        added call to SPHTRIG_TRANSLB (VK; 2013feb)
        bug fixes (theta0,phi0=0) (VK; 2013mar)
```

B.187 RANDOMPOW [/stat]

```
function        randompow
        return a sample drawn from a power-law distribution

        let the power-law be defined as
          f(x)=N*x^(-A)
        the cumulative function,
          F(X)= (N/(1-A))*(X^(1-A)-Xmin^(1-A))  if A.ne.1
              = N*alog(X)-N*alog(Xmin)          if A.eq.1
        which may be normalized to go from 0 to 1:
          G(X)=(X^(1-A)-Xmin^(1-A))/(Xmax^(1-A)-Xmin^(1-A))     if A.ne.1
              =(alog(X)-alog(Xmin))/(alog(Xmax)-alog(Xmin))     if A.eq.1
        given a random number 0<r==G(X)<1, solve for X:
          X=(r*(Xmax^(1-A)-Xmin^(1-A))+Xmin^(1-A))^(1/(1-A))    if A.ne.1
           =Xmin*exp(r*alog(Xmax/Xmin))                         if A.eq.1

syntax
        x=randompow(seed,num,alpha=alpha,/cumul,/outpoi,Nerr=Nerr,$
        Xmin=Xmin,Xmax=Xmax,/help,verbose=verbose)

parameters
        seed    [INPUT; necessary] seed for random number generator
                * this parameter must be present, but there is no
                  need for it to be defined on input.
                * will be passed w/o modification to RANDOMU()
        num     [INPUT] this has a different meaning depending on
                whether the keyword CUMUL is set or not.
                * if CUMUL is not set, this is the number of random
                  samples that are returned, with a minimum of 1
                * if CUMUL is set, this is taken to be the sum of the
                  numbers sampled, with a default of max{1,XMIN}

keywords
        alpha   [INPUT] index of power-law
                * default is 1.8
        cumul   [INPUT] if set, generates random samples until the
                sum of the values matches NUM.  otherwise, the sum
                is irrelevant and NUM samples are extracted
        outpoi  [INPUT] if set, a Poisson deviate is obtained from
                the sample generated
                * if CUMUL is set, the Poisson deviates are what are
                  summed to match against NUM
        Nerr    [INPUT] error on NUM.  The actual SUM{X} matches a random
                realization of NCT
```

```
                * if given, assumed to be 1-sigma Gaussian error
                * if not given, is set to sqrt(NUM+0.75)+1
        Xmin    [I/O] minimum value in sample
                * default: 2
        Xmax    [I/O] maximum value in sample
                * default: max{Normal(NUM,NERR),XMIN}
        help    [INPUT] if set, prints out usage and exits
        verbose [INPUT] controls chatter

        _extra  [JUNK] here only to prevent crashing the program

example
        .run randompow

history
        vinay kashyap (Nov09; based loosely on samplpl.pro)
        allowed XMIN,XMAX to be defined on return, modified example
          calls to be self-consistent (VK; JanMMX)
        allowed large photon counts by using N(mu,sqrt(mu)) to take
          over as Poisson (VK; JanMMXIII)
```

B.188 RD_APED [/mkemis]

```
function        rd_aped
        read in a line list from APED and return the 3D
        emissivities [WVLS,TEMPS,EDENS]

syntax
        emiss=rd_aped(apedfil,wvls,temps,edens,Zs,ions,levs,$
        apeddir=apeddir,xtnames=xtnames)

parameters
        apedfil [INPUT; required] FITS file, output of APEC
                * assumed to be in following format --
                  [PARAMETERS][cols kT,EDensity,Nelement,Nline]
                        where kT is in [keV], EDensity is in [cm^-3]
                  [L{logT}_{logNe}][cols Lambda,Lambda_Err,Epsilon,Epsilon_Err,Element,I
                        where {logT} and {logNe} are in '(f3.1)' or '(f4.1)',
                        Lambda* are in [\AA], and Epsilon* are in [ph cm^3/s]
        wvls    [OUTPUT; required] all the wavelengths listed in APEDFIL [\AA]
        temps   [OUTPUT; required] all the temperatures in APEDFIL [log[K]]
        edens   [OUTPUT; required] all the e-densities in APEDFIL [cm^-3]
        Zs      [OUTPUT; required] atomic numbers corresponding to each WVLS
        ions    [OUTPUT; required] ionic state of each of ZS
        levs    [OUTPUT; required] level designations for each line [2,N(WVLS)]

keywords
        apeddir [INPUT] directory containing APEDFIL
                * if not set, looks for APEDFIL in $cwd
        xtnames [OUTPUT] all the extension names
        _extra  [JUNK] here only to prevent crashing the program

restrictions
        for APED format as of Apr 2000 only
        subroutines: KILROY

history
        vinay kashyap (AprMM)
```

B.189 RD_CHANDRA_GEOM [/]

```
function        rd_chandra_geom
        reads in the entire geometry file from Chandra's CALDB
        into an IDL structure and spits it out

syntax
        geom=rd_chandra_geom(/help,caldb=caldb,version=version,verbose=verbose)

parameters      NONE

keywords
        help    [INPUT] if set, prints out the usage help
        caldb   [INPUT] top level directory containing the Chandra CALDB
                * default is to look for environment variable CALDB
                * if $CALDB is not defined, then assumes /soft/ciao/CALDB/
        version [INPUT] version number to use
                * default is 5
                * the geometry file is assumed to be at
                  $CALDB/data/chandra/tel/bcf/geom/telD1999-07-23geomN####.fits
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (Oct04)
        changed default VERSION to 5 (VK; Jul05)
```

B.190 RD_CHIANTI [/mkemis]

```
function        rd_chianti
        one stop function to read in all relevant atomic parameters from
        the CHIANTI v4 database and return them in a comprehensive structure
        (see structure variable CHELP for complete description).

usage
        line_database=rd_chianti(Z,ion,delec=delec,chidir=chidir)

parameters
        Z       [INPUT; required] atomic number
        ION     [INPUT] ionic state [if 0 or not given, assumes
                fully ionized, i.e., =Z]
                * Z and ION must be integer scalars

keywords
        delec   [INPUT] if set, look for the directory containing
                dielectronic recombination data (1/0)
        chidir  [INPUT] path to top directory of CHIANTI installation
                * default: !CHIDIR
                * hardcoded default: /data/fubar/SCAR/CHIANTI/dbase/
        _extra  [INPUT] junk -- ignore

restrictions
        * requires procedure SETSYSVAL
        * requires CHIANTI v4 or something very similar
        * requires PoA subroutine INICON
          and CHIANTI subroutines READ_WGFA2, READ_ELVLC, READ_SPLUPS, READ_IP

history
        written by vinay kashyap (Nov96), based on SYNTHETIC.PRO of CHIANTI
        added keyword DELEC, modified default behavior of ION (VK; OctMM)
        modified for CHIANTI v4, incompatible with earlier versions, added
        call to SETSYSVAL (VK; Jun02)
        force a "/" at the end of CHIDIR (VK/LwL; Sep02)
        added call to READ_IP and output ionization potentials via CHLINE,
          corrected multiplicity for DR lines (VK; Apr04)
```

B.191 RD_COCO [/mkemis]

```
function        rd_coco
        reads in the ATOMDB continuum emissivities and returns it in
        an IDL structure

syntax
        cocostr=rd_coco(cocofil,wrange=wrange,nbin=nbin,abund=abund,$
        /noerg,/noang,cemis=cemis,wvl=wvl,wmid=wmid,logT=logT,$
        verbose=verbose)

parameters
        cocofil [INPUT; required] full path to the location of the
                continuum emissivities database file
                * if set to '$COCO', looks for /data/atomdb/apec_coco.fits
                * if set to '$oldAPED', looks for /data/fubar/SCAR/atomdb_1.3.1/apec_coco.fits
                * if set to '$APED', looks for /data/fubar/SCAR/atomdb_v2.0.0/apec_coco.fits

keywords
        wrange  [INPUT] 2-element array to define the range over which
                the continuum is extracted
                * default is [0.01,1300]
        nbin    [INPUT] number of bins in the output array
                * if -ve, the bins will be in log scale, with longer
                  widths at longer wavelengths
                * default is 2000
        abund   [INPUT] if supplied, replaces the hardcoded abundances
                with new ones
                * if not set, keeps it as Anders & Grevesse
        noerg   [INPUT] if set, refrains from converting the units
                from [ph ...] to [erg ...]
        noang   [INPUT] if set, refrains from converting the units
                from [... /keV] to [... /Ang]
        cemis   [OUTPUT] emissivity array of size (TEMPERATURE,WAVELENGTH)
                collated from all the various extensions in COCOFIL,
                interpolated onto a standard wavelength grid, and with
                PoA style units [1e23 ergs cm^3/s/Ang] (unless NOERG and
                NOANG are set)
        wvl     [OUTPUT] the standardized output wavelength grid
        wmid    [OUTPUT] the mid-bin values of the wavelength grid
        logT    [OUTPUT] the temperature grid over which CEMIS is defined
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

history
```

```
vinay kashyap (Apr2009)
changed default $APED to point to v2.0.0 (VK; May2011)
```

B.192 RD_CONT [/]

```
function        rd_cont
        routine to read in the output of WRT_CONT_* and return continuum
        emissivities [1e-23 erg cm^3/s/Ang] for specified criteria

syntax
        emis=rd_cont(root,pres=pres,logP=logP,n_e=n_e,wrange=wrange,$
        dbdir=dbdir,abund=abund,metal=metal,wvl=wvl,logT=logT,fcstr=fcstr,$
        chdwvl=chdwvl,/twoph,/noff,/nofb,cocofil=cocofil,/xlogT)

parameters
        root    [INPUT; required] prefix of files containing the emissivities
                * if set to 'chianti', will carry out a real-time computation
                  of the continua using CHIANTI routines FREEFREE, FREEBOUND,
                  and TWO_PHOTON
                * if set to 'apec', will read in the emissivities from the
                  APEC continuum database table, which includes both regular
                  continuum and the so-called pseudo-continuum
keywords
        pres    [INPUT; default: 1e15] pressure [cm^-3 K]
        logP    [INPUT; default: 15] log(pressure [cm^-3 K])
                * LOGP takes precedence over PRES
                * if no match is found, uses nearest available dataset
        n_e     [INPUT] electron density [cm^-3].  if set,
                * overrides LOGP
                * appends a "D" to DBDIR
        wrange  [INPUT; default: all] wavelength range
                * if 2-element array, wrange==[wmin,wmax]
                * if 1-element array or scalar, wrange==[wr(0),wr(0)]
        dbdir   [INPUT; default: "cont"] directory to look for files in
                * $<whatever> : look for environment variable.  if missing,
                  following special case is hardcoded --
                  -- $CONT : see variable CONT_DB
                * ignored when ROOT='chianti' or ROOT='apec'
        abund   [INPUT; default: anders & grevesse] abundances
        metal   [INPUT; default: 1] metallicity
                * assumes ABUND to be of metallicity 1
                * if set, scales abundances from ABUND appropriately
        wvl     [OUTPUT] bin beginning wavelengths and final ending
                wavelength [Angstroms]
        logT    [OUTPUT] log10(Temperatures [K])
        fcstr   [OUTPUT] if specified in call on input, will contain all
                the output variables
                        {CONT_INT,LOGT,WVL,REH,TKEV,EKEV,MIDWVL}
```

```
                in that order in a structure.
        chdwvl  [INPUT] bin size to use for ROOT='chianti'
                * default is 0.005 Angstrom
       twoph    [INPUT] set this to _include_ two-photon continuum
       noff    [INPUT] set this to _exclude_ free-free continuum
       nofb    [INPUT] set this to _exclude_ free-bound continuum
                * TWOPH, NOFF, NOFB are ignored unless ROOT='chianti'
        cocofil [INPUT] full path to the location of the APEC
                continuum emissivities file
                * default is to use '$APED', which points to
                  /data/atomdb/apec_coco.fits
                * used only if CEROOT points to APEC
        xlogT   [INPUT] if set, returns the emissivity in whatever
                native logT grid as is in the databases
                * the default is to rebin it with REBINX() and put
                  it on the grid logT=findgen(81)*0.05+4
        verbose [INPUT] controls chatter
        _extra  [INPUT ONLY] pass defined keywords to RD_IONEQ(),
                FREE_FREE_CH, FREE_BOUND_CH, and TWO_PHOTON_CH.

restrictions
        * WRT_CONT_* must have been run first
        * cannot handle logT arrays that change from file to file
        * requires subroutines
          INICON
          GETABUND
          SYMB2ZION [LAT2ARAB]
          SETSYSVAL [LEGALVAR]
         RD_IONEQ
         FREE_FREE_CH  [ITOH_CH,SUTHERLAND_CH]
         FREE_BOUND_CH
         TWO_PHOTON_CH [RD_CHIANTI]
          RD_COCO
          GETPOADEF
          from CHIANTI: READ_IONEQ, READ_GFFGU, BILINEAR, READ_IP,
            READ_KLGFB, CONCAT_DIR, GET_IEQ, FREEBOUND_ION, ZION2FILENAME,
            FILE_EXIST, DATATYPE, FILE_STAT, DATA_CHK, SINCE_VERSION,
            READ_FBLVL, IS_DIR, EXIST, GET_DELIM, OS_FAMILY, TAG_EXIST,
            IDL_RELEASE, VERNER_XS, KARZAS_XS


history
        vinay kashyap (Dec97; modified from RD_LINE)
        allowed dbdir to be "$CONT" or something that may be read in
          via an environmental variable (VK; Dec98)
```

```
added missing file close statement (VK; 1999Feb)
bug was preventing reading nearest available database (VK; Mar99)
bug in last bug correction, wasn't reading element files (VK; Apr99)
allowed FCSTR output if set in command line (VK; AugMM)
changes suggested by Antonio Maggio: *_DB defaults (VK; NovMM)
added byte-order check; added calls to SWAP_ENDIAN() as suggested
  by Antonio Maggio; changed CONT_DB to CONT_CDB environment
  variable; added call to SETSYSVAL; now deciphers default
  emissivity directory location from where this file is located
  (VK; FebMMI)
added keyword VERBOSE (VK; Aug01)
corrected bug where first bin was being chopped off (VK; Jan04)
check to see whether input abundances are already ratios (VK; Apr04)
added option to use chianti routines to compute continuum, added
  TWO_PHOT keyword to toggle inclusion of two photon continuum when
  calculating via CHIANTI to save time (LL; Feb05)
added NOFF and NOFB keywords to toggle inclusion of CHIANTI free-free
  and free-bound continuum, allowed system variable !XUVTOP to be set
  if not present (LL; Jun05)
changed keyword TWO_PHOT to TWOPH, added keyword CHDWVL, and
  cleaned up interface and internals (VK; Jun05)
bug fix: $CONT/cont was missing the "/" on some systems (VK; May06)
added option to read in APEC_COCO files and added keyword XLOGT
  (VK; Apr09)
fixed complication with abund being input as ratio (VK; Nov13)
added call to GETPOADEF (VK; Aug15)
```

B.193 RD_IONEQ [/]

```
function        rd_ioneq
        return the values of the ionization equilibrium for ions
        of element Z at specified temperatures as ARRAY(NLOGT,NIONS,NZ).

        right now this is simply a wrapper to the CHIANTI routine
        READ_IONEQ.  someday other formats will be supported.

syntax
        ioneq=rd_ioneq(z,logt,eqfile=eqfile,chidir=chidir,zlost=zlost,$
        verbose=verbose)

parameters
        z       [INPUT; required] atomic number
                * must be integers
        logt    [INPUT/OUTPUT] 1D array of log10(temperatures [K]) at which to
                provide ionization equilibrium values.  if not defined, returns
                the temperatures at which the values in EQFILE are given.

keywords
        eqfile  [INPUT] pathname, relative to CHIDIR, of file containing
                ionization equilibrium values
                * default: !IONEQF
                * hard-coded default: ioneq/chianti.ioneq
        chidir  [INPUT] path name to the CHIANTI installation
                * default: !CHIDIR
                * hard-coded default: /data/fubar/CHIANTI/dbase
        zlost   [OUTPUT] average number of lost electrons at each temperature
                * will be of size (NLOGT,NZ) or (NLOGT)
        verbose [INPUT] controls chatter
        _extra  [INPUT] junk -- here only to prevent program from crashing

description
        reads in ionization equilibrium values a la CHIANTI
        and interpolates to appropriate temperatures

        for example, the ionization fraction of Fe XVII would be
                (rd_ioneq(26,logT))[*,17-1]
        or for that matter,
                (rd_ioneq([8,26,10],logT))[*,17-1,1]

requires
        SETSYSVAL
        READ_IONEQ (from CHIANTI)
```

```
        GETPOADEF

history
        vinay kashyap (Nov96; based on SYNTHETIC.PRO)
        added keyword _EXTRA (VK; Dec96)
        interpolation in log to avoid stepped look (VK; Feb97)
        smooth after interpolation to avoid segmented look (VK; Apr97)
        bug: force maximum IONEQ to be 1, not 10^(1)! (VK; 99Sep)
        added keyword VERBOSE, now takes CHIDIR and EQFILE defaults
          from !CHIDIR and !IONEQF if they are defined; added call to
          SETSYSVAL (VK; Jul01)
        cleaned up the interface, and allowed Z to be an array
          (VK; Jun02)
        added keyword ZLOST, converted "()" to "[]" (VK: Apr06)
        allowed EQFILE to be a plain filename; changed hardcoded
          defaults of EQFILE and CHIDIR (VK; Jul13)
        changed default of EQFILE to ioneq/chianti.ioneq (VK; Aug13)
        added call to GETPOADEF (VK; Aug15)
```

B.194 RD_LINE [/]

```
function        rd_line
        supremely flexible routine to read in the output of WRT_LINE
        and return line cooling emissivities [1e-23 erg cm^3/s] for
        specified criteria in an array of size [NLOGT,NWVL]

syntax
        emis=rd_line(atom,pres=pres,logP=logP,n_e=n_e,wrange=wrange,$
        dbdir=dbdir,wvl=wvl,logT=logT,z=z,ion=ion,jon=jon,src=src,$
        desig=desig,econf=econf,fstr=fstr,/allah,/help,verbose=verbose)

parameters
        atom    [INPUT] element whose line intensities are to be read.
                * can specify ionic state (e.g., 'FeXX')
                * may be an array (e.g., ['He', 'c 5', 'N V', 's_8'])
                * default -- ALL

keywords
        pres    [INPUT; default: 1e15] pressure [cm^-3 K]
        logP    [INPUT; default: 15] log(pressure [cm^-3 K])
                * LOGP takes precedence over PRES
                * if no match is found, uses nearest available dataset
        n_e     [INPUT] electron density [cm^-3].  if set,
                * overrides LOGP
                * appends a "D" to DBDIR
                * NOTE: PRES, LOGP, and N_E are initalized in the
                  system variables !GASPR, !LOGPR and !EDENS.
                  HOWEVER, these system variables are *not* used as
                  the defaults; these keywords must be explicitly set
                  in order to take effect.
        wrange  [INPUT; default: all] wavelength range
                * if 2-element array, wrange==[wmin,wmax]
                * if 1-element array or scalar, wrange==[wr[0],wr[0]]
        dbdir   [INPUT; default: !LDBDIR; hard-coded default: "$CHIANTI"]
                line database directory
                * if an array, only the first element is used
                * WARNING: if N_E is set, a "D" is appended at the end (but
                  after stripping any pre-existing "D"s, mind!)
                * $<whatever> : look for environment variable.
                  some special cases are:
                  -- $SPEX : from environment variable SPEX_LDB
                  -- $CHIANTI : from environment variable CHIANTI_LDB
                  -- $APED : from environment variable APED_LDB
                  -- $SCAR : from environment variable SCAR_LDB
```

```
                if the environmental variable is undefined, then some
                special cases are hardcoded:
                -- SPEX : TOPDIR/emissivity/spex
                -- CHIANTI : TOPDIR/emissivity/chianti
                -- APED : TOPDIR/emissivity/aped
                -- SCAR : TOPDIR/emissivity/xinti
        wvl     [OUTPUT] wavelengths [Angstroms]
        logT    [OUTPUT] log10(Temperatures [K])
        z       [OUTPUT] atomic numbers of elements generating given WVL
        ion     [OUTPUT] ionic state of element generating given WVL
        jon     [OUTPUT] the ionic state that matters, as far as ion-balance
                is concerned
        src     [OUTPUT] reference to source of line info
                * 1: SPEX, 2: CHIANTI, 3: APED
        desig   [I/O] level designations for lower & upper level
                for each transition
                * read in only if DESIG is present in call
        econf   [I/O] e-configurations for lower & upper level
                * read in only if ECONF is present in call
        fstr    [OUTPUT] if present in the calling sequence, will contain all
                the output variables
                        LINE_INT,LOGT,WVL,Z,ION,DESIG,ECONF,SRC,JON
                in that order in a structure.
        allah   [INPUT] if set, assumes all files are available in correct
                forms, and does no checking.
                * quicker, but UNFORGIVING
                * faith is not an option
        help    [INPUT] if set, prints out the calling sequence and quits
        verbose [INPUT] higher the number, greater the chatter
                * default is !VERBOSE
        _extra  [INPUT] junk -- here only to avoid crashing the program

restrictions
        * WRT_LINE must have been run first
        * requires subroutines
          SYMB2ZION
          LAT2ARAB
          INICON
          SETSYSVAL
          IS_KEYWORD_SET
          GETPOADEF
        * cannot (as yet) handle logT arrays that change from atom to atom

usage examples
        * f=rd_line(atom,pres=pres,logP=logP,wrange=wrange,dbdir=dbdir,$
```

```
            wvl=wvl,logT=logT,z=z,ion=ion,/allah)
        * f=rd_line('He',logP=20,wr=[200.,500.],wvl=wvl,logT=logT,desig=desig,$
          econf=econf,fstr=fstr,/allah)
        * f=rd_line(wr=16.776,wvl=wvl,logT=logT,z=z,ion=ion,desig=desig)
        * econf=1 & f=rd_line(wvl=wvl,logt=logt,z=z,ion=ion,econf=econf,/allah)
          oo=sort(abs(wvl)) & surface,alog10(f[*,oo]+1),logt,abs(wvl[oo]),/ylog

history
        vinay kashyap (Nov96)
        added keywords ECONF and FSTR, changed I/O (VK; Dec96)
        corrected bug in reading in SRC (VK; Jan96)
        added keyword N_E (VK; Jun97)
        corrected bug that was returning nonsense labels when Z+ION was
          specified (VK; Jul97)
        now handles trailing "/"s and/or "D"s correctly; accepts "0"
          as synonym for all atoms (VK; Jun98)
        accepts '$SPEX', '$CHIANTI', and '$SCAR' as hardcoded default
          values for DBDIR (VK; Oct98)
        accepts DBDIR defined via environmental variable; trapped DBDIR
          being input as vector; added keyword HELP (VK; Nov98)
        "All" is accepted as an atom (VK; Dec98)
        "_"s in DBDIR causing incorrect selections: corrected
          added keyword JON, added read to ATOM_jon (VK; 1999May)
        added keyword VERBOSE, allowed input of the form "ALL ion" (VK; MarMM)
        added /SWAP_IF_LITTLE_ENDIAN to OPENR calls (VK; AprMM)
        added $APE to list of automatic *_DB (VK; AprMM)
        changed to full compatibility with IDL5+ ("()"->"[]"); allowed
          read in of DESIG, ECONF and output of FSTR if present in call;
          bug correction if ATOM goes past Zn (VK; AugMM)
        CHIANTI 3.0 has occassional NaNs in emissivity -- put in hack to
          avoid them (VK; SepMM)
        changes suggested by A.Maggio: strmid params, *_DB defaults
          (VK; NovMM)
        changed *_DB to *_LDB environment variables; if ALLAH was set,
          DBDIR was not being prepended (VK; DecMM)
        added call to SETSYSVAL; now deciphers default emissivity directory
          location from where this file is located; added byte-order
          check; added calls to SWAP_ENDIAN() as suggested by Antonio Maggio
          (VK; FebMMI)
        changed defaults for DBDIR, VERBOSE to read from !LDBDIR, !VERBOSE
          (VK; Jul01)
        updated for IDL5.6 keyword_set([0]) behavior change for vectors
          (VK; 20Mar2006)
        added call to GETPOADEF (VK; 05Aug2015)
        bug fix: compilation error (VK; 10Sep2015)
```

B.195 RD_LIST [/]

```
function        rd_list
        read in line emissivities for a list of specified lines and returns
        the database in a standard structure format (RD_LINE for details)

syntax
        lstr=rd_list(lnlist,/incieq,desig=desig,econf=econf,mapping=mapping,$
        dbdir=dbdir,brklst=brklst,sep=sep,prefix=prefix,comment=comment,$
        pres=pres,logP=logP,n_e=n_e,chifil=chifil,chidir=chidir,eqfile=eqfile,$
        eps=eps,verbose=verbose)

parameters
        lnlist  [INPUT; required] line list
                * must be a string scalar or array
                * if a line contains a "/" or "./" or ".." in the first field,
                  assumed to be a filename unless the "/" is followed by
                  another "/" ("//") or a "*" ("/*"), in which case it is
                  assumed to be a comment.
                * any line prefixed by "%", "#", ";", "/*", "*/", "//",
                  and PREFIX are assumed to be comments
                * either the file or the direct input must describe the
                  line as:-
                        "Z ION <sep> WAVE <sep> SOURCE <sep> DESCRIPTION"
                  where Z is the atomic symbol, ION is the ionic state,
                  WAVE is in [Ang], and SOURCE is DBDIR (see RD_LINE)
                * Z is required
                  ION is useful
                  WAVE, SOURCE, and DESCRIPTION are optional
                  but WAVE is required if SOURCE is given
                  and WAVE and SOURCE are required if DESCRIPTION is given
                * if multiple candidates are available, DESCRIPTION may be
                  used to choose from among them (see MAPPING)
                * <SEP> is a separator, usually a <tab>, but had better
                  not be any of "," "+-" "-" and "?"
                * WAVE may be (any combination of):-
                  exact match           -- "WVL"
                  inexact match         -- "WVL +- dW" or "WVL ? dW"
                  range                 -- "WMIN,WMAX" or "WMIN-WMAX"
                  enclosed in brackets  -- "()" "[]" "{}"
                * DESCRIPTION is basically treated as a set of <SP>-separated
                  keywords, and if a given field is preceded by a "!" any
                  matches based on that field would be _excluded_ (e.g.,
                  "!DR" would not match any DR line)
```

```
keywords
        incieq  [INPUT] if set, output emissivities will include ion-balance
        mapping [INPUT] this keyword controls what happens when an entry in
                LNLIST matches multiple instances of (Z,ION,WAVE) in the
                input database.
                * if not set, or set to 0, returns all the emissivities,
                  exactly as found, in separate rows,
                * but if not set and yet DESCRIPTION is set, then assumed
                  to be +ve
                * if set, and is +ve, then
                  -- looks through the DESCRIPTION field of LNLIST, and
                     finds the closest match in the database and returns
                     only that.
                  -- if "closest match" is ambiguous, then default is set
                     to the strongest of the closest matches, and the user
                     is asked to verify.
                  -- if DESCRIPTION is not set, or the given DESCRIPTION
                     succeeds in eliminating >all< the matches, then asks
                     which one (the default being the strongest line in
                     the set).
                * if set, and is -ve, then adds the emissivities of all
                  the extra lines to that of the strongest line in the
                  set.
                * NOTE: if "WVL?dW" is set, and multiple lines are found
                  in the database, MAPPING is overridden and the user is
                  asked to choose anyway.
        dbdir   [I/O] directory in which to look for files
                * will be used ONLY if not given in LNLIST
                * if not specified, *default* will be set at run-time to
                  whatever was used last
                * on output, will be a string array of the DBDIRs used
                  NOTE: this last feature will soon be eliminated -- use
                  keyword BRKLST instead
        sep     [INPUT] separator used to delineate fields in LNLIST
                * the default is to separate the fields with several blanks
                  (<tab>)
        prefix  [INPUT; default='*'] any line beginning with this letter
                is considered a comment
                * this is in addition to "%", "#", ";", "/"
        comment [OUTPUT] any commented lines are returned in this variable
                as a string array
        brklst  [OUTPUT] structure containing a breakdown of all the
                fields in LNLIST, in the form
                        {SPECIES, WMIN, WMAX, DBDIR, DESCRIPTION}
        eps     [INPUT; default=1e-5] a small number
```

```
                * used as a "round-off" error for wavelength precision
                  if the WAVE field does not specify a range
        verbose [INPUT; default=0] controls chatter
        _extra  [INPUT ONLY] use this to pass defined keywords to
                -- RD_LINE:
                   N_E         electron density [cm^-3], default is 1e9
                               _explicitly set to 0_ to use PRES/LOGP
                   PRES        pressure [cm^-3 K]
                   LOGP        alog10(pressure [cm^-3 K])
                   DESIG       to read in level designations, set by default
                   ECONF       to read in e configurations, set by default
                -- FOLD_IONEQ:
                   CHIFIL      to read in from CHIANTI style database
                   EQFILE      name of ion balance file
                -- RD_IONEQ:
                   CHIDIR      path to CHIANTI installation

restrictions
        syntax incompatible for IDL <5
        function calls incompatible for idl <5.3
        requires subroutines:
          RD_LINE, SYMB2ZION, LAT2ARAB, FOLD_IONEQ, RD_IONEQ, READ_IONEQ,
          CAT_LN, LINEFLX, GETABUND, RDABUND, SYZE, INICON, IS_KEYWORD_SET

history
        vinay kashyap (Jun98)
        added keyword COMMENT; changed behavior of SETASK (VK; Oct98)
        DBDIR now contains all directories on output (VK; Nov98)
        changed input to FOLD_IONEQ from ION to JON (VK; 99May)
        allowed unfound elements to be read in also, up to a point (VK; 99Jul)
        added keyword MAPPING, enhanced LNLIST, deleted SETASK, converted
          to IDL5.3 (VK; MarMM)
        now quits if IDL v <5.3 (VK; DecMM)
        force MAPPING=1 if DESCRIPTION is set; added keyword BRKLST as
          first step towards eliminating DBDIR as output; added keyword
          VERBOSE and EPS; changed DESIG/ECONF behavior to be set by default
          (VK; JanMMI)
        changed default behavior to read in from constant-density database
          at N_E=1e9 if nothing is specified (VK; Jan'03)
        added ability to exclude fields via DESCRIPTION (VK; Apr'03)
        changed default SEP to "any series of one or more space or tab
          characters" (VK; Feb'04)
        changed SEP back to <tab> (VK; Sep'05)
        updated for IDL5.6 keyword_set([0]) behavior change for vectors
          (VK; 20Mar2006)
```

B.196 RD_OGIP_RMF [/util]

```
function        rd_ogip_rmf
        reads in the response matrix and associated axes from OGIP-compliant
        response file and returns everything in a structure of the form:

        {NNRG, ELO, EHI, NCHAN, EMN, EMX, N_GRP, F_CHAN, N_CHAN, MATRIX, FIRSTCHAN}

        where ELO and EHI refer to range of photon energies at which the
        response is valid, EMN and EMX refer to bin boundaries for each
        channel, N_GRP refers to number of groups of non-zero data, F_CHAN
        refer to beginning indices of channels, N_CHAN are the number of
        channels in each group, and the output response MATRIX excludes
        zeros to save space.  The value of FIRSTCHAN indicates whether F_CHAN
        indices are 0-based or 1-based.

        http://legacy.gsfc.nasa.gov/docs/heasarc/caldb/docs/memos/cal_gen_92_002/cal_gen
        for a description of the format

        WARNING: the EBOUNDS extension of RMFs, which defines EMN and EMX,
        should never be used directly for scientific purposes.  They
        contain the nominal energy boundaries for the PHA, PI, or pixel
        bins that define the MATRIX.  They are never exact values, and
        are useful only for plotting purposes.  It is the detector bins
        and their indices that are the be-all and end-all of RMFs.

syntax
        rmstr=rd_ogip_rmf(rmfile,effar=effar,eqvar=eqvar,spcar=spcar,$
        rmfimg=rmfimg,fchcol=fchcol,/shift1,verbose=verbose)

parameters
        rmfile  [INPUT; required] response matrix file

keywords
        effar   [OUTPUT] returns the effective area as a function of ELO
                * if an RMF file is read instead of an RSP file, EFFAR are
                  uniformly 1, unless it is ASCA in which case it is the QEs
                * EFFAR are calculated ONLY if needed.  i.e., the keyword
                  must be present in call (IDL5+) or defined on input (IDL5-)
        eqvar   [OUTPUT] returns an equivalent effective area in each
                detector channel, measuring the contribution to that
                channel from all photon energies.
        spcar   [OUTPUT] a variation on EQVAR, the convolution of a really
                flat intensity spectrum [ph/.../keV] with the RSP.
                In other words, what a flat spectrum would look like.
```

```
                * BEWARE: EQVAR and SPCAR are _not_ true effective area.
                  EFFAR are in _energy_ bins.  EQVAR and SPCAR are in
                  _channel_ bins.
                * CAVEAT VSVATOR!
                * I will deny ever having said this, but if you _must_
                  use one of these fake effective areas, use SPCAR.
        rmfimg  [OUTPUT] returns the RMF matrix as an image of size
                N(CHANNEL) x N(ENERGY)
                * computed only if this keyword is present (must also
                  be defined to be non-zero in IDL versions prior to 5.0)
        shift1  [SEMI OBSOLETE] some RMFs have position indices that are
                0-based and some are 1-based.  this keyword used to specify
                which was what.  however, this information should be in the
                FITS header in the keyword TLMIN{FCHCOL}, and that is where
                the program takes its value from.
                * BEWARE: if SHIFT1 is set _and_ TLMIN{FCHCOL} comes out as 0
                  (usually if it is not defined, or perhaps it really is 0)
                  then FIRSTCHAN is taken to be 1, and vice versa.
                * in other words, if set, overrides whatever the file
                  itself is saying
        fchcol  [INPUT] column number of F_CHAN in RMFILE
                * default is 4
        _extra  [JUNK] here only to prevent crashing the program

restrictions
        requires IDLASTRO library routine MRDFITS

history
        vinay kashyap (Apr99; modified from RDRESP.PRO)
        added keyword SHIFT1 (VK; JanMMI)
        slight modification in defining IDL version (VK; FebMMI)
        added keyword FCHCOL, overrode SHIFT1, added FIRSTCHAN to output
          structure (VK; Nov2001)
        commented out the "L5_MATRIX" lines (VK; Dec2001)
        added keywords EQVAR, SPCAR, and VERBOSE (VK; Jul02)
        bug correction: rspfil variable changed to RMFILE (LL/VK;Aug04)
       added ability to handle non-OGIP Astro-E2 XIS rmf file (LL/VK:Aug04)
        tweaked to speed up calculation of effar (VK; Dec04)
        added keyword RMFIMG (VK; May06)
```

B.197 RD_PIMMS_FILE [/util]

```
procedure       rd_pimms_file
        read in the effective area values from PIMMS file

syntax
        rd_pimms_file,pimmfl,effar,nrgar,ecol=ecol,acol=acol,/wave

parameters
        pimmfl  [INPUT; required] scalar string specifying the full
                pathname to PIMMS effective area file.
                * output of GET_PIMMS_FILE
        effar   [OUTPUT] effective areas, usually in [cm^2]
        nrgar   [OUTPUT] energies at which EFFAR are defined
                * assumed to be [keV]
                * if WAVE is set, converted to [Ang]

keywords
        acol    [INPUT; default=2] column number which contains EFFAR
        ecol    [INPUT; default=1] column number which contains NRGAR
        wave    [INPUT] if set, assumes that ECOL contains [keV] and
                converts to [Ang]
                * if columns are ALREADY in [Ang], DO NOT SET THIS KEYWORD
        _extra  [JUNK] ignore -- here only to prevent crashing the program

subroutines
        STR_2_ARR, WC, STR_SEP, SYZE

history
        vinay kashyap (Mar99)
        changed call to STR_2_ARR from STR2ARR (VK; AprMMV)
```

B.198 RDABUND [/util]

```
function        rdabund
        return abundance values read from file

        disk file must have one entry per line, with either atomic symbol
        or atomic number denoting the element (or both), and the actual
        abundance also.  neither the order of the columns nor the order
        of the rows matter.  the fields must be separated by "SEP", usually
        a <tab>.  missing elements are filled out with values from Anders &
        Grevesse (see GETABUND).

syntax
        abund=rdabund(filnam,comment=comment,sep=sep,inorm=inorm,norm=norm)

parameters
        filnam  [INPUT; required] name of file containing the abundances

keywords
        comment [OUTPUT] normally, any line prefixed by ";" "%" "#" or "/"
                are ignored.  these lines are stored as a string array and
                output via this variable
        sep     [INPUT] single-character field separator
                * if set to "." "+" "-" or any of the alphanumerics, is ignored
                * if not set, tries <sp>, <tab>, "," and ":" in sequence
        inorm   [INPUT] format of the abundances in the input file
                * see GETABUND for description
                * if INORM is defined, first converted to n(Z)/n(H), then
                  converted back to whatever NORM is set to
        norm    [INPUT] exactly as in GETABUND
        _extra  [JUNK] here only to prevent crashing the program

subroutines
        GETABUND
        SYZE

history
        vinay kashyap (Dec98)
```

B.199 RDARF [/util]

```
function        rdarf
        reads in the spectral response and associated axes from
        OGIP-compliant ancillary response file

syntax
        arf=rdarf(arffil,ostr,units=units)

parameters
        arffil  [INPUT; required] name of ARF
        ostr    [OUTPUT] structure containing the axes information for the
                response: {NNRG, ELO, EHI}
                * ELO and EHI refer to range of photon energies at which
                  the response is valid

keywords
        units   [OUTPUT] contains the units of ARF; typically 'cm**2'
        _extra  [INPUT] junk -- here only to prevent crashing the program

restrictions
        requires IDLASTRO library routine MRDFITS

history
        vinay kashyap (Jul97)
        exits with 0 if ARFFIL not in fits format (VK; Apr05)
```

B.200 RDB [/util]

```
function        rdb
        return a structure containing the output of ascii table
        in RDB format

        RDB format is assumed to be:
        a bunch of comments preceded by "#"s in first column
        a header stating the names of each column
        a format description specifying the type of the column
        (S for strings, N for numeric)
        and each column is separated by a "<tab>"

        some special things understood by this program are:
        -- blank lines are treated as comments
        -- different symbols (e.g., "%") may be used as comment characters
        -- output comes out as double or float or long depending on what is
           in the first row
        -- double precision may be forced by prepending a ">" to "N"

syntax
        tab=rdb(filename,cols=cols,comm=comm,fldsep=fldsep,rdbdir=rdbdir,$
        verbose=verbose)

parameters
        rdbfil  [INPUT; required] name of rdb file
                * the ".rdb" extension is not necessary
                * a trailing "." is removed, so if there's a file without
                  an extension, set RDBFIL to "filename."
                * and if there's a file named "filename.", set RDBFIL to
                  "filename.." and so on.

keywords
        cols    [INPUT] string or integer array of columns to extract
                * if string, column names must match *exactly*, but
                  case is not important
                * if integers, remember that this is IDL, so first column
                  is indexed as 0
                * if /COLS is set, returns all columns, so if you want
                  just the second column, remember to say COLS=[1] and
                  not COLS=1
        comm    [INPUT] comment character
                * default is a "#"
        fldsep  [INPUT] field separator
                * default is a <TAB>
```

```
        rdbdir  [INPUT] if given, prepends RDBFIL with this pathname
        verbose [INPUT] if set, blabbers incoherently along the way
        _extra  [JUNK] here only to prevent crashing the program

subroutines
        STR_2_ARR
        CREATE_STRUCT
        KILROY
        LEGALVAR

restrictions
        works only on UNIX

history
        vinay kashyap (MIM.XII)
        illegal column names are now checked for (VK; MM.I)
        bug: preceding spaces turned values into zeros; converted all
          array subscripting to IDL5 (VK; JulMM)
        changed call to STR_2_ARR from STR2ARR (VK; MMV.IV)
        now looks at only the first character of format column field
          to decide type (VK; MMXIII.VIII)
        now basically returns only long integer and doubles (VK; MMXIII.XII)
```

B.201 RDPHA [/util]

```
function        rdpha
        return a structure containing the variables in an XSPEC style
        PHA file.  (doesn't really have to be a PHA file.  any FITS
        table file will do.)

parameters
        infile  [INPUT; required] name of PHA file

keywords
        ext     [INPUT] if set, only reads in the specified extension
        _extra  [JUNK] here only to avoid crashing the program

restrictions
        * requires the IDLASTRO library
        * empty extensions are treated as EOFs
        * can only decipher tables, not images
        * cannot handle Type II PHA files

history
        vinay kashyap (Jan MM)
```

B.202 RDRESP [/util]

```
function        rdresp
        reads in an OGIP-compliant response matrix file and returns the
        full response matrix

syntax
        rsp=rdresp(rspfil,ostr,effar=effar,fchcol=fchcol,/shift1)

parameters
        rspfil  [INPUT; required] name of response file
        ostr    [OUTPUT] structure containing all the information for the
                response matrix: {NNRG, ELO, EHI, NCHAN, EMN, EMX, N_GRP,
                F_CHAN, N_CHAN, MATRIX, FIRSTCHAN}
                (see RD_OGIP_RMF() for description)
                * output response matrix is in the form RSP(NNRG,NCHAN)

keywords
        effar   [OUTPUT] returns the effective area as a function of ELO
                NOTE: if an RMF file is read instead of an RSP file,
                EFFAR are actually the efficiencies
        _extra  [INPUT ONLY] pass defined keywords to subroutines
                RD_OGIP_RMF: FCHCOL,SHIFT1

restrictions
        requires IDLASTRO library routine MRDFITS
        requires RD_OGIP_RMF

history
        vinay kashyap (Jul97)
        added keyword SHIFT1 (VK; JanMMI)
        removed SHIFT1, added call to RD_OGIP_RMF, cleaned up (VK; Nov2001)
```

B.203 READ_ELVLC [/external]

```
PROJECT:  CHIANTI

      CHIANTI is an atomic database package for the calculation of
      astrophysical emission line spectra.  It is a collaborative project
      involving Ken Dere (Naval Research Laboratory, Washington DC),
      Brunella Monsignori-Fossi and Enrico Landi (Arcetri Observatory,
      Florence), and Helen Mason and Peter Young (DAMTP, Cambridge Univ.).


NAME:
       READ_ELVLC

PURPOSE:

       to read files containing observed and theoretical energy levels

CATEGORY:

       science.

CALLING SEQUENCE:

      READ_ELVLC, File, L1, Term, Conf, ss, ll, jj, Ecm, Eryd, Ecmth, Erydth, Ref


INPUTS:

       File:   the name of the file
              i.e., !xuvtop+'/si/si_12/si_12.elvlc' for Si XII

OPTIONAL INPUTS:

       None:


OUTPUTS:       L1      - level index
               Term    - configuration index
               Conf    - configuration description
               ss      - 2S+1
               ll      - L
               jj      - J
               Ecm     - observed energy (cm^-1)
```

```
               Eryd    - observed energy (Rydbergs)
               Ecmth   - theoretical energy (cm^-1)
               Erydth  - theoretical energy (Rydbergs)
               Ref     - reference

  note:  the theoretical energies are usually those used in the scattering
         calculation and are only useful for predicting approximate
         wavelengths


EXAMPLE:

              > file = !xuvtop+'/si/si_12/si_12.elvlc'
              > read_elvlc,file,l1,term,conf,ss,ll,jj,ecm,eryd,ref
              >

MODIFICATION HISTORY:
       Written by:     Ken Dere
       March 1996:     Version 2.0

      Ver.2, 6-Dec-2001, Peter Young
         Modified the form of the "term" output. The J-value is now
         represented by a fraction rather than a decimal.
```

B.204 READ_FBLVL [/external]

```
PROJECT:  CHIANTI

      CHIANTI is an Atomic Database Package for Spectroscopic Diagnostics of
      Astrophysical Plasmas. It is a collaborative project involving the Naval
      Research Laboratory (USA), the University of Florence (Italy), the
      University of Cambridge and the Rutherford Appleton Laboratory (UK).


NAME:
       READ_FBLVL

PURPOSE:

       To read files containing observed free-bound energy levels

CATEGORY:

       Science.

CALLING SEQUENCE:

      READ_FBLVL, File, L1, Conf, pqn, ll, spd, mult, Ecm, Ecmth, Ref


INPUTS:

       File:   the name of the file
              e.g., !xuvtop+'/si/si_12/si_12.fblvl' for Si XII

OPTIONAL INPUTS:

       None:


OUTPUTS:       L1    - level index
               CONF  - configuration description
               PQN   - principal quantum number of electron being ionized
               LL    - L-value of ionized electron
               SPD   - 'S', 'P', etc to denote L value
               MULT  - multiplicity  2J+1
               ECM   - energy (cm^-1)
               ECMTH - energy (cm^-1)
```

```
               REF   - reference

EXAMPLE:

            > file = !xuvtop+'/si/si_12/si_12.elvlc'
            > read_fblvl,file,l1,conf,pqn,ll,mult,ecm,ecmth,ref
            >

HISTORY

     Ver.1, 31-Jul-2002, Peter Young
```

B.205 READ_GFFGU [/external]

```
PROJECT:  CHIANTI

      CHIANTI is an atomic database package for the calculation of
      continuum and emission line spectra from astrophysical plasmas. It is
      a collaborative project involving the Naval Research Laboratory
      (Washington DC, USA), the Arcetri Observatory (Firenze, Italy), and the
      Cambridge University (United Kingdom).

NAME:
      READ_GFFGU

PURPOSE:

      Read gffgu.dat file containing free-free gaunt factors of
            R. S. Sutherland, 1998, MNRAS, 300, 321
            note:  the file available from the web site are mislabelled


CALLING SEQUENCE:

      READ_GFFGU,g2,u,gff


INPUTS:

      None


OUTPUTS:

      g2,u,gff defined in the paper by Sutherland



COMMON BLOCKS:

      None



MODIFICATION HISTORY:
      Written by:     Ken Dere
```

```
April 2000:     Version 3.0
October 2001:   Version 4 - Ken Dere -
                corrected for the fact that the original
                Sutherland file was mislabelled
November 2001:  Corrected the address of the gffgu.dat file - Enrico Landi
```

B.206 READ_IONEQ [/external]

```
PROJECT:  CHIANTI

      CHIANTI is an Atomic Database Package for Spectroscopic Diagnostics of
      Astrophysical Plasmas. It is a collaborative project involving the Naval
      Research Laboratory (USA), the University of Florence (Italy), the
      University of Cambridge and the Rutherford Appleton Laboratory (UK).


NAME:
       READ_IONEQ

PURPOSE:

       to read files containing the ionization equilibrium values

CATEGORY:

       science.

CALLING SEQUENCE:

      READ_IONEQ, File, T, Ioneq, Ref


INPUTS:

       File:   for example, !xuvtop+'/ioneq/arnaud_rothenflug.ioneq'

OPTIONAL INPUTS:

       None:


OUTPUTS:

       T:  array of log10 temperatures
      Ioneq:  3-d array (T,element,ion)
              of the fractional abundance of the ion in
             ionization equilibrium.
      Ref:  reference in the scientific literature
```

```
EXAMPLE:

            > read_ioneq,!xuvtop+'/ioneq/arnaud_rothenflug.ioneq'
            > help,t,ioneq
            > T               FLOAT     = Array(41)
            > IONEQ           FLOAT     = Array(41, 28, 29)
            > print, minmax(t)
            >  4.00000      8.00000
            > print,t(20)
            >  6.0000
            > print,ioneq(20,25,9)
            >   0.269                  = fractional abundance of  Fe X in
                                         ionization equilibrium

MODIFICATION HISTORY:
       Written by:     Ken Dere (KPD)
       March 1996:     Version 2.0
      March 1999:     KPD to read both number of temperature and number
                      of elements

      25-Oct-2000     V. 4, Giulio Del Zanna (GDZ).
                      Corrected to interpret the '-1' as a reference only
                      if within the first 3 columns

      V.  5, 21-May-2002, Giulio Del Zanna (GDZ)
                  generalized directory concatenation to work for
                  Unix, Windows  and VMS.

      v.6, 25-Oct-2004, Peter Young
           modified format statement so that it will read any number of
           temperatures.

      V 7, 25-May-2005, GDZ
                 corrected routine header.

VERSION     :   7, 25-May-2005
       Aug2010:        Vinay Kashyap commented out reference to !xuvtop
```

B.207 READ_IP [/external]

```
PROJECT:  CHIANTI

      CHIANTI is an atomic database package for the calculation of
      continuum and emission line spectra from astrophysical plasmas. It is a
      collaborative project involving the Naval Research Laboratory
      (Washington DC, USA), the Arcetri Observatory (Firenze, Italy), and the
      Cambridge University (United Kingdom).


NAME:
       READ_IP

PURPOSE:

       to read values of ionization potentials


CALLING SEQUENCE:

      READ_IP, File, IP, Ref


INPUTS:

       File:  the name of the file containing the IP values, usually
              !xuvtop/ip/chianti.ip


OUTPUTS:

       IP:  Array values of ionization potential (cm^-1)
      Ref:  the reference to the IP values in the scientific literature





EXAMPLE:

            > read_ip,!xuvtop+'/ip/chianti.ip',ip,ref
                 ip(2-1,2-1) give the ionization potential of He II  (Z=2, Ion=2)
```

```
MODIFICATION HISTORY:
       Written by:     Ken Dere
       March 1998:     Version 1.0
```

B.208 READ_KLGFB [/external]

```
PROJECT:  CHIANTI

      CHIANTI is an atomic database package for the calculation of
      continuum and emission line spectra from astrophysical plasmas. It is a
      collaborative project involving the Naval Research Laboratory
      (Washington DC, USA), the Arcetri Observatory (Firenze, Italy), and the
      Cambridge University (United Kingdom).


NAME:
       READ_KLGFB

PURPOSE:

       to read CHIANTI files file containing the free-bound gaunt factors for n=1-6
      from Karzas and Latter, 1961, ApJSS, 6, 167

CATEGORY:

       science.

CALLING SEQUENCE:

      READ_KLGFB,PE,GFB


INPUTS:

       File:  none




OUTPUTS:

       PE:   an array of natural log of 41 values of Photon Energy values relative to
            the recombination edge i.e., at the recombination edge, PE=0.
      GFB:  an array containing the natural log of free-bound gaunt factors indexed
            by energy, n and l
```

```
 PROCEDURE:  reads the file:  '!xuvtop/continuum/klgfb.dat'



EXAMPLE:

            > read_klgfb,pe,klgfb
                 the free-bound gaunt factor vs. energy for recombination to a hydrogen
                 (n=2 and l=1) is given by klgfb(*,n-1,l-1)

MODIFICATION HISTORY:
       Written by:     Ken Dere
       July 2002:     Version 1.0
            Version 2, 8-Aug-02, GDZ
            corrected a typo.

VERSION     :   V.2, 8-Aug-02
```

B.209 READ_MASTERLIST [/external]

```
PROJECT:  CHIANTI

      CHIANTI is an atomic database package for the calculation of
      continuum and emission line spectra from astrophysical plasmas. It is a
      collaborative project involving the Naval Research Laboratory
      (Washington DC, USA), the Arcetri Observatory (Firenze, Italy), and the
      Cambridge University (United Kingdom).


NAME:
       READ_MASTERLIST

PURPOSE:

       read the masterlist.ions types of file and output a list of ions

CATEGORY:

       science.

CALLING SEQUENCE:

      READ_MASTERLIST,filename,mlist


INPUTS:

       filename:   name of the masterlist file



KEYWORD PARAMETERS:

       none

OUTPUTS:

       mlist:  list of ions



COMMON BLOCKS:
```

```
       none;


EXAMPLE:

            > read_masterlist,'!xuvtop/masterlist.masterlist.ions',mlist

MODIFICATION HISTORY:
       Written by:     Ken Dere
       December 1998:  first version

      Ver.2, 23-Sep-2010, Peter Young
        - renamed LIST to MLIST for compatibility with IDL 8
```

B.210 READ_SPLUPS [/external]

```
PROJECT:  CHIANTI

      CHIANTI is an atomic database package for the calculation of
      continuum and emission line spectra from astrophysical plasmas. It
      is a
      collaborative project involving the Naval Research Laboratory
      (Washington DC, USA), the Arcetri Observatory (Firenze, Italy), and the
      Cambridge University (United Kingdom).


NAME:
       READ_SPLUPS

PURPOSE:

       to read file containing spline fits to the Burgess-Tully scaled
      collision strengths

CATEGORY:

       science.

CALLING SEQUENCE:

      READ_SPLUPS, File, Splstr, Splref


INPUTS:

       File:   the name of the input file, i.e. !xuvtop/si/si_4/si_4.splups


OUTPUTS:

      SPLSTR  Structure containing the data from the file. The tags are
              as follows:

              .lvl1   lower level index
              .lvl2   upper level index
              .t_type transition type
              .gf     gf value
              .de     Delta-E for transition (rydbergs)
```

```
              .c_ups  the scaling parameter
              .nspl
              .spl    Vector of length 9, containing spline points


OPTIONAL OUTPUTS

      SPLREF  String array containing references.

KEYWORDS

      PROT    Allows reading of .psplups files for proton rates.


PROCEDURE:

       see Burgess and Tully, 1992, Astronomy and Astrophysics, 254, 436.

EXAMPLE:

      > read_splups, !xuvtop+'/si/si_4/si_4.splups',splstr,splref

PROGRAMMING NOTES

      This routine is marginally quicker (20-25%) reading the .splups
      files than Ken's original routine. The improvement in speed is
      through minimising the lines of code in the WHILE loop.

MODIFICATION HISTORY:
       Written by:     Ken Dere
       March 1996:     Version 2.0

      Ver.3, 23-Jan-01, Peter Young
               completely revised. Now reads into a structure and
               handles 9 point spline fits.

      Ver.4, 26-Jan-01, Peter Young
               speeded up routine

      Ver.5, 22-Mar-01, Peter Young
               now checks if file exists

      Ver.6, 17-Jul-2012, Ken Dere
              can handle files with 5 through 9 spline points
```

B.211 READ_WGFA2 [/external]

```
PROJECT:  CHIANTI

      CHIANTI is an atomic database package for the calculation of
      continuum and emission line spectra from astrophysical plasmas. It is a
      collaborative project involving the Naval Research Laboratory
      (Washington DC, USA), the Arcetri Observatory (Firenze, Italy), and the
      Cambridge University (United Kingdom).


NAME:
       READ_WGFA2

PURPOSE:

       read radiative data files
        a modified version of read_wgfa
        needed to take account of two types of transitions between the same 2 levels:
        for example, the M1 and the 2 photon E1 transition 1s-2s in hydrogenic ions


CATEGORY:

       science

CALLING SEQUENCE:

      READ_WGFA2, File, Lvl1, Lvl2, Wvl, Gf, A_value, Ref


INPUTS:

       File:  name of the file containing the radiative data
               i.e. !xuvtop/c/c_4/c_4.wgfa


OUTPUTS:

       Lvl1:  1D array of indices of the lower level (starting at 1)
      Lvl2:  1D array of indices of the upper level (starting at 1)
      Wvl:   1D array of transition wavelengths in Angstroms
      Gf:    1D array of weighted oscillator strength gf
      A_value:  1D array of the total radiative transition probability (s^-1)
```

```
      Ref:   1D string array of references to the data in the scientific literature



EXAMPLE:

            > read_wgfa2,!xuvtop+'/c/c_4/c_4.wgfa',lvl1,lvl2,wvl,gf,a,ref


MODIFICATION HISTORY:
       Written by:     Ken Dere
       March 1996:     Version 2.0

      V.3, 3-Nov-03, Giulio Del Zanna (GDZ)
       Modified size of input arrays. Now the routine can read a file up to
       30 000  lines.

      V.4, 26-May-09, Enrico Landi (EL)
       Modified size of input arrays. Now the routine can read a file up to
       100,000 lines.
       Modified definition of variable index (now index=0L).

      V.5, 26-May-09, Peter Young
       Modified the way references are read.


VERSION     :  5, 26-May-09
```

B.212 REBINW [/util]

```
function        rebinw
        rebins the input array on a new grid and returns the output array.
        this routine is preferred over simple INTERPOL or SPLINE because
        this is guaranteed to e.g., conserve flux while rebinning spectra.

syntax
        ff=rebinw(f,x,y,/perbin,numbins=numbins,nbin=nbin,wrange=wrange,$
        /slowok,verbose=verbose)

parameters
        f       [INPUT; required] array values
        x       [INPUT; required] absissae for F
                1: if size matches that of F, assumed to be mid-bin values
                2: if size is size(F)+1, assumed to be bin boundaries
                   (i.e., all the bin-beginners and a final bin-ender)
                3: if neither of the above, this is assumed to be the
                   desired *output* grid, and the absissae of the input
                   are assumed to span the linear range 1..N(F)
                   * Y is ignored on input, but will be overwritten on output
        y       [I/O] the new absissa values, taken from NUMBINS if it is set.
                1: if scalar or 1-element vector on input, assumed to be
                   number of bins in output
                   * linear binning if +ve, log binning if -ve
                2: if >1-element vector on input, assumed to be the desired
                   output grid of bin boundaries, with the last element
                   defining the final bin-ending value.
                3: if not defined, takes value from NBIN
                NOTE: F extrapolates as zeros.

keywords
        perbin  [INPUT] if set, assumes that units of F(X) are [.../bin]
        nbin    [I/O] number of bins in output
                * overwritten if defined via X or Y
                * linear binning if +ve, log binning if -ve
                * default: 1
        numbins [INPUT] same as NBINS, except that if set, it
                overrides NBIN _and_ Y
                * so, just to reiterate:
                  if NUMBINS is _not_ defined, then
                  -- if Y is a scalar, say = NN, then
                        output will be rebinned to have NN elements, and
                        Y will be overwritten to contain the new grid, and
                        NBIN will be reset to NN
```

```
                  -- if Y is a vector, then
                        output will be rebinned using Y as bin boundaries, and
                        NBIN will be reset to N(Y)-1
                  -- if Y is not defined, but NBIN is defined, then
                        output will be rebinned to have NBIN elements, and
                        Y will on output contain the new grid
                  if NUMBINS is defined, then
                  -- output will be rebinned to have NUMBINS elements, and
                     Y will contain the new grid on output, and
                     NBIN will be reset to NUMBINS
        wrange  [INPUT] output grid range
                * ignored if defined via array Y (or X -- case 3)
                * overrides if determined via X and CASE 1 of Y
                * default: [1.,500.]
        slowok  [INPUT] if set, skips call to FINDEX and does it the
                slow IDL way
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

restrictions
        beware that if Y is not sorted in ascending order, on output
        it will be!
        requires external procedure FINDEX.PRO

history
        vinay kashyap (Jan98)
        forced x to be required inputs (Jun98)
        changed call to INTERPOL to combo call to INTERPOLATE/FINDEX (VK;99Jul)
        corrected long-standing "feature" of NBIN being useless, changed
          default behavior of Y=UNDEFINED, NBIN, and X=/=F (VK; MarMM)
        added keyword SLOWOK (VK; Jul01)
        added keyword NUMBINS (VK; Dec'02)
        added keyword VERBOSE; now uses /CUMULATIVE in TOTAL() (VK; Feb'03)
        beware of /CUMULATIVE in TOTAL(), which returns an array of same
            size as input, and does _not_ begin with 0 (VK; Apr'03)
        added warning if Y gets resorted (VK; Mar04)
        cleaned up behavior in the case of non-unique/non-monotonic inputs
          (VK; Mar05)
```

B.213 REBINX [/util]

```
function       rebinx
        returns the input array suitably reformatted (resampled, stretched,
        constricted, what have you) to the new range.

syntax
        f=rebinx(array,inX,outX,Xindex=Xindex,/absmin,/absmax,/norng,$
        verbose=verbose)

parameters
        array   [INPUT; required] array to be reformatted
        inX     [INPUT; required] abscissa values defining the input grid
        outX    [INPUT] abscissa values defining the output grid
                * if not given, assumed to be FINDGEN(81)*0.05+4.

keywords
        Xindex  [INPUT] specifies which dimension of ARRAY corresponds to inX
                * if not given, picks that dimension that first matches to the
                  number of elements in inX
                * e.g., if ARRAY[1000,81] and inX[81], then Xindex=1
        absmin  [INPUT] if set, force oARRAY to be > ABSMIN
                * default is min(ARRAY)
        absmax  [INPUT] if set, force oARRAY to be < ABSMAX
                * default is max(ARRAY)
        norng   [INPUT] if set, ignores ABSMIN and ABSMAX
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

example
        1. rebin DEM in range logT=5..7 in 11 steps to DEM in range
           logT=4..8 in 81 steps:
                fullDEM=rebinx(DEM,findgen(11)*0.2+5,findgen(81)*0.05+4)
        2. resample emissivity from full range to smaller sample:
                subEMIS=rebinx(EMIS,logT,sublogT,Xindex=0)

restrictions
        * can only handle at most 2-D arrays

history
        vinay kashyap (Oct98)
        bug correction with Xindex, added keyword VERBOSE, converted
          to IDL5 format (VK; Jul02)
```

B.214 REFITLINES [/scrypt]

```
REFITLINES.PRO
        so you ran through FITLINES and got your fits.  and you are
        ready to publish.
        but wait, did you use the right model?  sigh.
        ok, let's have a go at it again.

        need the spectrum, of course.  should be in variables
        LAMBDA and SPECT[LAMBDA] and SPECTSIG[LAMBDA]

        need the old fit structure.  should be in variable
        OLDFITSTR

        new fit structure will be in variable, what else, NEWFITSTR

vinay kashyap (MMJul)
        included HISTERIX (VK; MMIVFeb)
```

B.215 REGROUP [/util]

```
function        regroup
        group an input array of counts such that each group has
        at least a minimum number of counts.  start accumulating
        from one end, and keep going until the threshold is
        exceeded, then start on the next group, etc.

syntax
        gct=regroup(counts,minct,grid,/newgrid,iout=iout,verbose=vv)

parameters
        counts  [INPUT; required] counts array to be regrouped
        minct   [INPUT; required] minimum total counts in each group
        grid    [I/O] bin indices indicating starting and ending index
                of each group, such that group K runs from index
                GRID[K] to GRID[K+1]-1
                * if given as an array that contains indices from 0
                  to N_ELEMENTS(COUNTS) on input, uses that gridding
                  and does not calculate a new grid, unless NEWGRID
                  is set
                * if not well-defined, or if NEWGRID is set, will be
                  overwritten on output

keywords
        newgrid [INPUT] if set, ignores the input GRID and computes
                the grouping and the resulting new grid
                * if not set, and GRID is well-defined, GRID is used
                * if not set, and GRID is not well-defined, a new
                  GRID is computed and the input is overwritten
        iout    [OUTPUT] an integer index array designating which
                group each element of COUNTS has ended up in
                * this illustrates the essential non-destructive nature
                  of the regrouping -- it should always be possible to
                  go back to the original array and rearrange it.
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

example
        y=randomu(seed,20) & gy=regroup(y,2.5,x,/newgrid,iout=iout)
        print,y,iout & print,gy,x
        for i=0,n_elements(gy)-1 do print,i,x[i],x[i+1]-1,$
          gy[i],total(y[x[i]:x[i+1]-1]),total(y[where(iout eq i)])

these are all the smoothing tools in PINTofALE
```

```
        ALALOESS() makes a loess curve
        CLTSMOOTH() removes -ves from background subtracted spectra or light curves
        CONV_RMF convolves with specified RMF
        HAARTRAN() smoothes by filtering on Haar wavelet coefficients
        LINEREM() removes lines from spectrum by iterative S/N filtering
        NOISMOOTH() does boxcar accumulation a la Ebeling's asmooth
        REGROUP() accumulates from one end
        SCRMF does fast convolution using optimized RMF
        SMOOTHIE does peak-descent accumulation to build up S/N
        SPLAC computes a segmented piecewise linear approximations
        UNKINK() removes sharp kinks from a curve
        VARSMOOTH() does local smoothing at specified scales
        VOORSMOOTH() does fast local smoothing at specified scales

history
        vinay kashyap (Oct09)
```

B.216 REGROUP_RMF [/util]

```
function   regroup_rmf

        Converts RMF structure created with  RD_OGIP_RMF() to a
        one group per photon energy RMF structure. Done so to
        maximize convolution efficiency with SCRMF() e.g. for
        model fitting via RMFs in FITLINES(). Will return input
        if RMF is already optimal. If effective area is folded
        into matrix and this is undesired, REGROUP_RMF is also
        capable of returning an unfolded RMF via the NOEFFAR
        keyword.

        If ENRG paramater is fed,  will be used to define the photon
        energy grid of the reconstructed RMF.

        If a spectrum SPEC is fed together with ENRG, then
        the photon energy range will be constructed so that
        bin-density is highest at energy ranges with largest
        signal.

        If NB is set, will be used as the total number of bins
        for resulting photon energy grid.

syntax  nrmfstr = regroup_rmf(rmfstr,enrg,spec,nb,/noeffar)

parameters
        rmfstr  [INPUT;required] structure containing info of
                response matrix, the output of RD_OGIP_RMF()
        enrg    [INPUT] desired energy grid
        spec    [INPUT] spectrum of interest.  if given, should
                 be on the same grid as ENRG, and is used to tweak
                 the energy grid in the output such that the
                 bin-density is correlated with SPEC.
                 * overrides ENRG except that the energy range in
                   the output is taken from ENRG.
        nb      [INPUT] total number of photon energy bins desired.
                 * is used only when SPEC is defined, and by default
                   is set to N(ENRG) if not given

keywords
        noeffar [INPUT] if set, the LSF corresponding to each
                photon energy will be normalized to 1 i.e.
                effective area is unfolded.
                 WARNING: Because photons are _lost_ at the edges
```

```
                 of applicability of the RMFs, the renormalization
                 will not necessarily be correct in these regions.
history
        liwei lin (Aug 03)
               ADDED option to regrid rmf in enrgy space via
                enrg, spec, and nb (LL Sep03)
               BUG FIX CF handling/interpolation re-worked to avoid
                shifting of grid (LL Sep03)
               BUG FIX now robust to zero vavlues for N_GRP.
                thanks Jan-Uwe (LL Jun06)
                BUG FIX changed condition for when regroup is done
                 thanks Jan-Uwe (VK Dec07)
```

B.217 RENORMOD [/fitting]

```
function        renormod
        given model values and data, find a normalization factor for
        the model that makes it a better fit to the data and return
        this new value.

syntax
        norm=renormod(data,model,err,/uchisq,/uabsdev,/umaxdev,/ureldev,$
        statval=statval,dstep=dstep,eps=eps,verbose=verbose)

parameters
        data    [INPUT; required] data values
        model   [INPUT; required] model values
                * size must match DATA
        err     [INPUT; optional] errors to use to determine the goodness
                of fit statistic.  if not given, assumed to be the Gehrels
                approximation to Poisson, sqrt(abs(DATA+0.75)+1)
                * size must match DATA, or must be 1-element
                * if 1-element and +ve, constant value errors are assumed
                * if 1-element and -ve, a constant fractional value for
                  the errors, DATA*abs(ERR) are assumed
                * WARNING: data points with ERR=0 are ignored when it
                  is necessary to divide by the errors

keywords
        uchisq  [INPUT] if set, use the chi-square statistic as the
                measure of goodness of fit
                * this is the default
        uabsdev [INPUT] if set, use abs(DATA-MODEL) as statistic
        umaxdev [INPUT] if set, use min(abs(DATA-MODEL)) as statistic
        ureldev [INPUT] if set, use total(abs(DATA-MODEL)/ERR) as statistic
                * priority: UCHISQ > UABSDEV > UMAXDEV > URELDEV
        statval [OUTPUT] "best-fit" value of the statistic
        dstep   [INPUT] value of the first step
                * default is 10% of initial value
        eps     [INPUT] a small number
                * default is 1e-6
        maxstep [INPUT] maximum number of iterations
                * default is 100
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to avoid crashing the program

history
        vinay kashyap (Jun2001)
```

B.218 ROI_SELEZIONI [/util]

```
function        roi_selezioni
        simple tool to pick out a set of pixels defining a region of
        interest in a 2D image; returns selection as 1D index array

syntax
        iroi=roi_selezioni(image,iseed,thresh=thresh,jitter=jitter,$
        hadrian=hadrian,verbose=verbose)

parameters
        image   [INPUT; required] image from which to select region of interest
        iseed   [INPUT] pixel indices to act as a seed selection
                * if given, will use these points as the selected
                  pixels and work in batch processing mode, i.e.,
                  will NOT be interactive

keywords
        thresh  [INPUT] if given, then filters the input image such
                that IMAGE > THRESH
                * if not given, assumed to be 0
        jitter  [INPUT] sensitivity in distinguishing between click and drag
                * default is 1.5 pixels
        hadrian [INPUT] thus far and no further.  if set, places a fixed
                boundary around the area selected with the cursor, i.e.,
                connected regions are not followed out of this area.  set
                this to pick out individual pixels, for instance.
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (Aug06)
        added parameter ISEED and keywords THRESH, HADRIAN _EXTRA;
          allowed selection extending outside cursor selected areas;
          changed name from pickoutroi to roi_selezioni (VK; May07)
```

B.219 ROOFN [/util]

```
function        roofn
        returns the smallest number greater than a specified number
        that is a power of the specified base.

        output will be of same type as BASE.

syntax
        nroof=roofn(num,base,floor=floor,pwr=pwr)

parameters
        num     [INPUT; required] number to "round up"
        base    [INPUT; default=2] base

keywords
        floor   [INPUT] if set, returns the largest integer LESS THAN M
                that is a power of N.
        pwr     [OUTPUT] the appropriate index [N^(PWR)]
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (April 1995)
        added keyword _EXTRA (VK; Mar99)
        cleaned up a bit (VK; Apr99)
        cleaned up some more (VK; Dec07)
        completely rewritten to be more flexible and faster (VK; Jan13)
```

B.220 RUNDENS [/util]

```
function        rundens
        compute and return the running density of a set of measurements.

syntax
        yrun=rundens(xpt,hwidth,/usex,/optwdt,outhwdt=outhwdt,$
        yerr=yerr,xerr=xerr,nsim=nsim,ysim=ysim,yesim=yesim,xtol=xtol,$
        wts=wts,dwts=dwts,lbound=lbound,rbound=rbound,cvmse=cvmse,$
        verbose=verbose)

parameters
        xpt     [INPUT; required] array of points for which the running density
                must be calculated
                * a running density is calculated at each value of XPT
        hwidth  [INPUT] the half-width on either side of a point to include in
                the density
                * this is assumed to be the number of array indices
                  unless USEX is set
                * default is to use 5% of n_elements(XPT)
                  or 5% of the range of XPT if USEX is set
                * if OPTWDT is set, computes the optimum width to use
                  and treats this as just a default
                * actual value used is returned via keyword OUTHWDT

keywords
        usex    [INPUT] if set, assumes HWIDTH is in same units as XPT
        optwdt  [INPUT] if set, figures out the optimum HWIDTH to use by
                comparing the width of the distribution of the running
                density estimates to the expected statistical stddev for
                a flat density
                * if this is set, HWIDTH will be overridden
                * the optimum HWIDTH is calculated based on the array
                  indices -- i.e., USEX is ignored during the calculation,
                  but if set, is used afterwards to scale the optimum
                  HWIDTH to the units in XPT.  So beware that it may not
                  really be the optimum in this case.
                * if this module fails, the smallest width checked will
                  be returned
        outhwdt [OUTPUT] will contain the value of HWIDTH actually used in
                the calculation
        yerr    [OUTPUT] error bar on the density at each point
                * WARNING: adjacent values of YERR are not independent.
                  while the error estimate at each point is fine, to
                  get a truly independent estimate, one has to skip by
```

```
                  2*HWIDTH
        xerr    [INPUT] error bars on XPT
                * used to compute a Monte Carlo uncertainty band on the
                  output running density
        nsim    [INPUT] number of simulations to carry out
                * default is none
        ysim    [OUTPUT] a NSIMxN(XPT) array to contain the running
                density for each simulation
        yesim   [OUTPUT] point-wise stddev based on YSIM
        xtol    [INPUT] the tolerance for considering two points to be
                identical
                * essentially, if any two adjacent points are found to
                  be identical, then all points are jittered by +-0.5*XTOL
        wts     [INPUT] if given, computes the total of the WTS and returns
                that rather than the running density
                * MUST match the size of XPT, otherwise ignored
        dwts    [INPUT] same as WTS, but the total is computed and placed
                in the denominator
                * MUST match the size of XPT, otherwise ignored
                * if both WTS and DWTS are given, the ratio of the totals
                  is computed, not the average of the ratios
        lbound  [OUTPUT] array indices of leftside bounds for all XPT
        rbound  [OUTPUT] array indices of rightside bounds for all XPT
        cvmse   [OUTPUT] cross-validation mean square error between estimate
                and leave-one-out estimate.  the extra calculations are
                performed only if this keyword is present and are not done
                for the simulated data.
        verbose [INPUT] controls chatter

        _extra  [JUNK] here only to prevent crashing the program

subroutines
        RUNDENS_OPTWDT (included here)
        CURVESECT
        EQT_INTERVAL

history
        vinay kashyap (MMXI.VII)
        added keywords WTS and DWTS and corrected bug (VK; MMXI.VIII)
        added keywords LBOUND, RBOUND, and CVMSE (VK; MMXII.V)
        added keyword OPTWDT, added function RUNDENS_OPTWDT, and added
          calls to RUNDENS_OPTWDT(), CURVESECT(), and EQT_INTERVAL()
          (VK; MMXII.XII)
```

B.221 SCALE_VECTOR [/external]

```
NAME:
      SCALE_VECTOR

PURPOSE:

      This is a utility routine to scale the points of a vector
      (or an array) into a given data range. The minimum value of
      the vector (or array) is set equal to the minimum data range. And
      the maximum value of the vector (or array) is set equal to the
      maximum data range.

AUTHOR:

      FANNING SOFTWARE CONSULTING
      David Fanning, Ph.D.
      1645 Sheely Drive
      Fort Collins, CO 80526 USA
      Phone: 970-221-0438
      E-mail: davidf@dfanning.com
      Coyote's Guide to IDL Programming: http://www.dfanning.com

CATEGORY:

      Utilities

CALLING SEQUENCE:

      scaledVector = SCALE_VECTOR(vector, minRange, maxRange)

INPUT POSITIONAL PARAMETERS:

      vector:   The vector (or array) to be scaled.
      minRange: The minimum value of the scaled vector. Set to 0 by default.
      maxRange: The maximum value of the scaled vector. Set to 1 by default.

INPUT KEYWORD PARAMETERS:

      DOUBLE:        Set this keyword to perform scaling in double precision.
                     Otherwise, scaling is done in floating point precision.

      MAXVALUE:      Any value in the input vector greater than this value is
                     set to this value before scaling.
```

```
      MINVALUE:      Any value in the input vector less than this value is
                     set to this value before scaling.

      NAN:           Set this keyword to enable not-a-number checking. NANs
                     in vector will be ignored.

      PRESERVE_TYPE: Set this keyword to preserve the input data type in the output.

RETURN VALUE:

      scaledVector: The vector (or array) values scaled into the data range.

COMMON BLOCKS:
      None.

EXAMPLE:

      x = [3, 5, 0, 10]
      xscaled = SCALE_VECTOR(x, -50, 50)
      Print, xscaled
         -20.0000     0.000000     -50.0000      50.0000

RESTRICTIONS:

    Requires the following programs from the Coyote Library:

       http://www.dfanning.com/programs/convert_to_type.pro
       http://www.dfanning.com/programs/fpufix.pro

MODIFICATION HISTORY:

      Written by:  David W. Fanning, 12 Dec 1998.
      Added MAXVALUE and MINVALUE keywords. 5 Dec 1999. DWF.
      Added NAN keyword. 18 Sept 2000. DWF.
      Removed check that made minRange less than maxRange to allow ranges to be
         reversed on axes, etc. 28 Dec 2003. DWF.
      Added PRESERVE_TYPE and DOUBLE keywords. 19 February 2006. DWF.
      Added FPUFIX to cut down on floating underflow errors. 11 March 2006. DWF.
```

B.222 SCRMF [/util]

```
procedure       scrmf
        Convolves input energy spectrum with the response matrix

       SCRMF is a version of CONV_RMF optimized for speed.
       It maximized speed at the expense of memory.
       A faster version of CONV_RMF was necessary for
       use with e.g., FITLINES() because the convolution procedure is
       called literally tens of thousands of times.

       On input, the RMF is required to be an RMF structure created
       by RD_OGIP_RMF() from an OGIP-compliant FITS file AND
       optimized with REGROUP_RMF().

       NOTE: If the RMF is not optimized prior to
       the SCRMF call, REGROUP_RMF() will be called
       internally, and SCRMF will be slowed to a crawl.

syntax
        scrmf,nrg,flx,chan,spec,rmf,effar=effar,nrgar=nrgar,$
        rmfstr=rmfstr,fchcol=fchcol,/shift1

parameters
        nrg     [INPUT; required] mid-bin values at which input FLX is defined.
               * preferably NRG is identical to RMF channel energy grid
                * depends on the RMF, but is invariably [keV]
                * if size = N(FLX)+1, assumed to be bin-boundaries
        flx     [INPUT; required] fluxes in [ph/bin/(s/sr/...)]
                * note that this assumes that the effective areas are
                  already folded in.  this assumption is made in order
                  to speed up the calculation some more
                * the input should also be integrated over the bin width,
                  i.e., do not give [ph/keV/...] or [ph/Ang/...] as input
                * also note, the units should not be [erg/...] -- be sure
                  to divide the input energy spectrum by the energy of
                  the photons corresponding to that bin before calling
                  the program
        chan    [OUTPUT; required] bin boundary values at which output SPEC
                is defined.
               * same energy range as NRG (so size will not necesarily match
                  EMN,EMX of RMF)
                * same units as RMF.EMN, RMF.EMX
        spec    [OUTPUT; required] output, convolved spectrum
        rmf     [I/O; required] rmf response structure in RD_OGIP_RMF() format.
```

```
                WARNING: Unlike CONV_RMF, this _cannot_ be the name of a file.
                Also, the input will be overwritten if necessary.
               NOTE: For most efficient use with FITLINES(), RMFs which
               are defined with only one group per photon energy bin are
               preferred. If RMF contains more than one group per
               photon energy, the RMF will be optimized for FITLINES()
               use with REGROUP_RMF().

keywords
       verbose [INPUT] set verbosity level
        effar   [INPUT] (not implemented yet)
        nrgar   [INPUT] (not implemented yet)
        rmfstr  [OUTPUT] structure containing info of response matrix,
                the output of RD_OGIP_RMF()
        _extra  [INPUT ONLY] pass defined keywords to subroutines

subroutines
        REGROUP_RMF

these are all the smoothing tools in PINTofALE
        ALALOESS() makes a loess curve
        CLTSMOOTH() removes -ves from background subtracted spectra or light curves
        CONV_RMF convolves with specified RMF
        HAARTRAN() smoothes by filtering on Haar wavelet coefficients
        LINEREM() removes lines from spectrum by iterative S/N filtering
        NOISMOOTH() does boxcar accumulation a la Ebeling's asmooth
        REGROUP() accumulates from one end
        SCRMF does fast convolution using optimized RMF
        SMOOTHIE does peak-descent accumulation to build up S/N
        SPLAC computes a segmented piecewise linear approximations
        UNKINK() removes sharp kinks from a curve
        VARSMOOTH() does local smoothing at specified scales
        VOORSMOOTH() does fast local smoothing at specified scales

history
      liwei lin  (Aug 03)
          BUGFIX crashed with no warning when input spectrum
          not fully covered. Thanks Jan-Uwe. LL (Jun 06)
          BUGFIX crashed with zero values for N_CHAN.
          Thanks Jan-Uwe. LL (Jun 06)
        BUGFIX crashed when FIRSTCHAN=1 for IBEG=0 (thanks Jan-Uwe; Vinay K; May 07)
```

B.223 SDSS2FLUX [/util]

```
function        sdss2flux
        convert SDSS asinh magnitudes to fluxes in ergs/s/cm^2/Hz

syntax
        fx=sdss2flux(mag,/gband,/rband,/iband,/zband,/uband,verbose=verbose)

parameters
        mag     [INPUT; required] SDSS (asinh) magnitude
                * could be scalar or array

keywords
        uband   [INPUT] input is u mags
        gband   [INPUT] input is g mags
        rband   [INPUT] input is r mags
        iband   [INPUT] input is i mags
        zband   [INPUT] input is z mags
                * default is GBAND
                * if multiple keywords are set,
                  GBAND > RBAND > IBAND > ZBAND > UBAND
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

description
        see SDSS photometric calibration documents
        http://www.sdss3.org/dr10/algorithms/fluxcal.php
        http://www.sdss3.org/dr10/algorithms/magnitudes.php#asinh
        http://classic.sdss.org/dr6/algorithms/fluxcal.html

example
        print, 'solar luminosity in g band = ',sdss2flux(4.7,/gband)*4.*!dpi*(3.1d18*10.

history
        vinay kashyap (2014sep16)
```

B.224 SETABUND [/util]

```
function        setabund
        widget-based editor to set abundances

syntax
        abund=setabund(init=init,norm=norm,/help)

parameters      NONE

keywords
        init    [INPUT] initial set of abundances -- default is Anders &
                Gervasse (1989)
        help    [INPUT] as it says
        norm    [INPUT] same as in GETABUND, to decide whether the
                abundances are in normal or log format.
                * abs(NORM) > 1 ==> log
        _extra  [INPUT] junk -- here only to prevent program from crashing

restrictions
        widget-based

subroutines
        GETABUND
        GETEMALL, SETEMALL (appended)

history
        vinay kashyap (Dec96)
        bug correction (MULT not being set) (VK; Jan96)
```

B.225 SETCOLOR [/obsolete]

```
procedure       setcolor
        loads specified colors into color table in specified indices

warning
        this routine is being renamed SETKOLOR to avoid conflict with
        JHUAPL routine of the same name.  this will disappear very soon,
        and is hanging around only to aid in the transition.

syntax
        setcolor,colors,icols,rgbfil=rgbfil,oldr=oldr,oldg=oldg,oldb=oldb

parameters
        colors  [INPUT; required] string array containing the names
                of the colors to be loaded
        icols   [INPUT] integer array specifying into which index
                each color goes to
                * if size does not match COLORS, ignored
                * default is to go from 0:255

keywords
        rgbfil  [INPUT] name of file containing (r,g,b) color values
                and corresponding names in same format as:-
                * the default, /usr/lib/X11/rgb.txt
        oldr    [OUTPUT] the old R colors
        oldg    [OUTPUT] the old G colors
        oldb    [OUTPUT] the old B colors
                * NOTE: to undo the setting, just do
                        TVLCT,OLDR,OLDG,OLDB
        _ref_extra      [JUNK] here only to prevent crashing the program

restrictions
        * RGBFIL bit works only on UNIX
        * requires subroutine NARY2DEC

history
        vinay kashyap (99Jul)
        bug correction for ICOLS=!D.N_COLORS
        STRMID must have 3 args for back-compatibility (VK FebMMI/A.Maggio)
        the bell tolls (VK; FebMMI)
```

B.226 SETCONT [/]

```
function        setcont
        interactively determine a continuum

syntax
        cont=setcont(x,y,ysig,ycont=ycont,xcont=xcont,/const,$
        xrange=xrange,yrange=yrange,ccol=ccol,/lock)

parameters
        x       [INPUT; required] abscissa values
                * must be an array with at least 2 elements
        y       [INPUT; required] Y(X)
                * length must match X
        ysig    [OUTPUT] errors on returned continuum (see description below)

keywords
        ycont   [I/O] if set on input, begins work from here
                * overwritten by new values on output
        xcont   [I/O] if N(YCONT) .NE. N(X), then look here for the
                actual abscissae values.. YCONT=YCONT(XCONT)
                * if N(XCONT) .NE. N(YCONT), both are ignored
        const   [I/O] if set, returns piecewise >constant< continum,
                as opposed to piecewise >linear< curve
                * can be toggled by clicking outside the plot area
        xrange  [INPUT] xrange of plot -- passed with no comment to PLOT
        yrange  [INPUT] yrange of plot -- passed with no comment to PLOT
        ccol    [INPUT] color of continuum line
                * default is 150*!D.N_COLORS/256
        lock    [INPUT] if set, locks up the relative positions of YCONT
                * has an effect ONLY if YCONT is set
                * side-effects:
                  -- does not increase number of N({XY}CONT)
                  -- left or middle click moves the entire curve up and down
        _extra  [JUNK] here only to prevent crashing the program

error calculation
        since the continuum is being set interactively, there is no meaningful
        way to define a statistical error on it.

restrictions
        works in X.  only?

history
        vinay kashyap (Oct98)
```

```
increased default plotting YRANGE (VK; Nov98)
added keyword LOCK (VK; Dec98)
cursor changes shape when mouse button is pressed (VK; FebMM)
allowed return of zero errors, cleaned up error handling, added CCOL
  cleaned up LOCK (VK; MarMM)
undid the cursor shape changes (VK; AprMM)
bug: wasn't holding the poisson v/s zero error toggle (VK; JulMM)
improved color-scale setting for 24-bit consoles (VK; FebMMI)
changed default to produce errors=0 rather than Poisson errors
  (VK; Aug03)
```

B.227 SETKOLOR [/util]

```
procedure       setkolor
        loads specified colors into color table in specified indices
        (this used to be called "setcolor" but the name was changed to
        avoid a clash with JHUAPL's procedure of the same name)

syntax
        setkolor,colors,icols,ocols,rgbfil=rgbfil,oldr=oldr,oldg=oldg,oldb=oldb

parameters
        colors  [INPUT; required] string array containing the names
                of the colors to be loaded
        icols   [INPUT] integer array specifying into which index
                each color goes to
                * if size does not match COLORS, ignored
                * default is to go from 0:255
        ocols   [OUTPUT] byte array of size [3,N(COLORS)] containing
                the hexadecimal values determined for each COLORS, as an
                rgb triplet

keywords
        rgbfil  [INPUT] name of file containing (r,g,b) color values
                and corresponding names in same format as:-
                * the default, /usr/lib/X11/rgb.txt
                * note: some new linux systems don't seem to have this file.
                  those users can copy over this file from another system, or
                  use the copy provided in !ARDB+'/rgb.txt'
        oldr    [OUTPUT] the old R colors
        oldg    [OUTPUT] the old G colors
        oldb    [OUTPUT] the old B colors
                * NOTE: to undo the setting, just do
                        TVLCT,OLDR,OLDG,OLDB
        _ref_extra      [JUNK] here only to prevent crashing the program

example
        plot,findgen(10)/10.,color=1 & setkolor,'green',1
        oplot,sin(findgen(10.)),color=2 & setkolor,'#ff0000',2

restrictions
        * requires subroutine NARY2DEC
        * RGBFIL bit works only on UNIX
        * messes with the current color table
        * feature, not a bug: if !D.N_COLORS < 256, and a SETKOLOR
          command is issued before a plot window is opened, a color
```

```
          table is loaded in which gets lopped off at !D.N_COLORS,
          and all subsequent plots appear dim.

history
        vinay kashyap (99Jul)
        bug correction for ICOLS=!D.N_COLORS
        STRMID must have 3 args for back-compatibility (VK FebMMI/A.Maggio)
        changed name from SETCOLOR to SETKOLOR (VK FebMMI)
        added optional output OCOLS and check for UNIX (VK; Jun02)
        added checks to find rgb.txt in !ARDB if, as in some linux
          distributions, /usr/lib/X11/rgb.txt doesn't exist (VK; Mar07)
        bug correction: local copy was trying to override RGBFIL (JJD; May08)
        added switch to choose file_search() instead of findfile()
          for IDL 7+ (JJD/VK; Apr14)
```

B.228 SETSYSVAL [/util]

```
procedure       setsysval
        sets a system variable (the kind preceded by a "!") to the
        specified value

syntax
        setsysval,sysv,val,/getval,verbose=v,/Read_Only

parameters
        sysv    [INPUT; required] a string scalar containing the
                name of the system variable, written out without the
                obligatory "!" (e.g., 'PATH')
                * if !SYSV does not exist, creates it using defsysv
                  inside an execute()
        val     [I/O] if defined on input, will set the value of
                !SYSV to this value, if !SYSV allows overwriting;
                if not defined, or is illegal (e.g., wrong type of
                input) then on output contains the value.
                * if GETVAL is set, then regardless of all above considerations
                  VAL contains the value of !SYSV on _output_

keywords
        getval          [INPUT] if set, returns the existing value of
                        !SYSV in VAL
        verbose         [INPUT; default=0] controls chatter
        Read_Only       [INPUT] if set, defines new !SYSV as read only
        _extra          [JUNK] here only to prevent crashing the program

subroutines
        LEGALVAR

history
        vinay kashyap (OctMM)
```

B.229 SHOW_LINE [/]

```
procedure       show_line
        plot lines and label them

syntax
        show_line,wvl,flux,Z=Z,ion=ion,lambda=lambda,spec=spec,order=order,$
        wmark=wmark,fmark=fmark,lmark=lmark,cmark=cmark,xo=xo,yo=yo,oxr=oxr,$
        oyr=oyr,sep=sep,/squish,dynrng=dynrng,markp=markp,markc=markc,$
        marks=marks,marko=marko,/quiet,xtitle=xtitle,ytitle=ytitle,$
        title=title

parameters
        wvl     [INPUT; required] wavelengths of lines
        flux    [INPUT; optional] fluxes of lines

keywords
        Z       [INPUT] atomic number of element producing WVL
        ion     [INPUT] ionic state of element producing WVL
                * Z and ION do not have to match the size of WVL:
                  as many are used as available or necessary
        spec    [INPUT] spectrum over which to plot the lines
        lambda  [INPUT] as in SPEC(LAMBDA)
                * SPEC and LAMBDA should match in size, or else they
                  are both ignored
        order   [INPUT] string of comma-separated integers denoting
                grating orders to include in plot
                * default is to use just the first order
                * anything meaningless (including "0") defaults to first order
        wmark   [INPUT] additional locations to mark (the output of SMUDGE,
                for example)
        fmark   [INPUT] fluxes at which to mark WMARKs
        lmark   [INPUT] labels for WMARKs
                * FMARK and LMARK are ignored unless they match WMARK
        cmark   [INPUT] colors for LMARKs
        markc   [INPUT] colors for WVLs
                * default is 222
        xo      [OUTPUT] output from PICKRANGE
        yo      [OUTPUT] output from PICKRANGE
        oxr     [OUTPUT] output from PICKRANGE
        oyr     [OUTPUT] output from PICKRANGE
        _extra  [INPUT] pass defined keywords to subroutines:-
                STR_2_ARR: SEP,SQUISH
                PICKRANGE: DYNRNG,MARKP,MARKS,MARKO,QUIET,XTITLE,YTITLE,TITLE
```

```
subroutines
        STR_2_ARR
        INICON
        PICKRANGE

history
        vinay kashyap (Nov98)
        added keywords WMARK, FMARK, LMARK (VK; Dec98)
        changed call to INITSTUFF to INICON (VK; 99May)
        added keywords MARKC and CMARK (VK; JanMMI)
        bug fix: if FLUX not set, 1st 2 were being marked at [0,1] (VK; Feb01)
        changed call to STR_2_ARR from STR2ARR (VK; AprMMV)
```

B.230 SINCE_VERSION [/external]

```
Project     :  SOHO - CDS

Name        :
       SINCE_VERSION

Purpose     :
       Determine if current release is later than specified.

Explanation :
       Determine if the current release of IDL (as given in the
       !VERSION.RELEASE system variable) comes after the user specified
       release.

Use         :
       test = SINCE_VERSION( release )

       Use the /FTOXDR keyword to the BYTEORDER procedure if the current
       release of IDL is 2.2.2 or later

       IDL> if since_version('2.2.2') then byteorder, a, /FTOXDR

Inputs      :
       release - scalar string, must be formatted exactly like the
               !VERSION.RELEASE system variable (e.g. '3.0.0')

Opt. Inputs :
       None.

Outputs     :
       test - 1 if current release is identical or later than the specified
             'release' else 0

Opt. Outputs:  None.

Keywords    :  None.

Calls       :  None.

Common      :  None.

Restrictions:  None.

Side effects:  None.
```

```
Category    :  Utilities, Miscellaneous

Prev. Hist. :
       Written   Wayne Landsman         Hughes/STX        January, 1992
       Corrected algorithm     W. Landsman                April, 1992

Written     :  Wayne Landsman, Hughes/STX, January 1992

Modified    :  Version 1, William Thompson, GSFC, 14 December 1994
                       Incorporated into CDS library

Version     :  Version 1, 14 December 1994
```

B.231 SKYPOS [/util]

```
procedure       skypos
        process input RA and Dec and produce the appropriate output

syntax
        skypos,ra,dec,out1,out2,/i24,/osxg,/hrs,/idnum,idpre=idpre,$
        idform=idform,/PM,chop=chop,osep=osep,verbose=verbose

parameters
        ra      [INPUT; required] Right Ascension
                * may be a real or string array
                * if real, assumed to be decimal degrees unless
                  the keyword I24 is set
                * if string, will check whether decimal or sexagesimal
                  -- sexagesimal if ’:’ or ’ ’ exist, decimal otherwise
        dec     [INPUT; required] Declination
                * size _must_ match RA
                * may be real or string array
                * if string, will check whether decimal or sexagesimal
                  -- sexagesimal if ’:’ or ’ ’ or ’     ’ exist, decimal otherwise
                NOTE: all calculations are performed in double precision
        out1    [OUTPUT; required] output, depends on keyword (see below)
        out2    [OUTPUT; optional] output, depends on keyword (see below)
                * by default,
                  OUT1 = RA [decimal deg] and OUT2 = Dec [decimal deg]
                * if keyword IDNUM is set, then
                  OUT1 = IAU style ID number = OUT2
                  -- in addition, keyword IDFORM overrides keywords
                     OSXG and HRS (see below)
                * if keyword OSXG is set, then
                  OUT1 = RA [h:m:s] and OUT2 = Dec [d:m:s]
                * if keyword HRS is set, then
                  OUT1 = RA [decimal hours] and OUT2 = Dec [decimal deg]

keywords
        i24     [INPUT] set this to indicate whether the input RA is
                in decimal hours, not degrees
        osxg    [INPUT] if set, output RA and Dec will be in sexagesimal
                units, not decimal
        hrs     [INPUT] if set, output RA will be in hours
                (in sexagesimal if OSXG is set, in decimal otherwise)
                * if neither OSXG nor HRS is set, then RA will be
                  in decimal degrees
        idnum   [INPUT] if set, an IAU style name is constructed and
```

```
                returned as a string in OUT1
        idpre   [INPUT] a uniform prefix to prepend to IDNUM
                * default is "J"
        idform  [INPUT] a string denoting the format for how IDNUM
                is written out
                * default is 'HMS.sXDMS'
                * ignored if IDNUM is not set
                * format rules:
                  -- always has RA first and Dec next
                  -- X denotes the sign of the declination (optional)
                  -- RA is specified either as
                        "R.r" [deg] or "HMS.s" [sexagesimal]
                  -- Dec is specified either as
                        "D.d" [deg] or "DMS.s" [sexagesimal]
                  -- repetions of small letters after the period denote the
                     number of digits, e.g., "R.rrr" is written as f7.3
                  -- repetitions of capital letters before the period has
                     no effect unless they are greater than the minimum
                     required, e.g.,
                     "R.r" == f5.1 == "RR.r" == "RRR.r", but "RRRR.r" == f6.1
        PM      [INPUT] if set, replaces "+/-" with "P/M" in the sign of Dec
        chop    [INPUT] chops off the specified number of digits off the
                right hand end of the IDFORMatted output
        osep    [INPUT] set the separator between fields in a sexagesimal
                output (see keyword OSXG) to OSEP rather than ":"
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

subroutines
        DEG2HRS
        STR_2_ARR
        SYZE

history
        vinay kashyap (Jul03)
        bug correction when output was R.rrr, and added keyword PM (VK; Dec03)
        added keyword OSEP (VK; May04)
        updated RA and Dec outputs to ss.ssss by default and changed
          all internal calcs to double precision (VK; Aug15)
```

B.232 SMOOTHIE [/util]

```
procedure       smoothie
        return the input function smoothed such that a minimum required S/N
        is met at each point in the function.  useful for smoothing out the
        weak wings and continuua in line dominated spectra.  starts from
        the peak of the lines and steps down the profile, accumulating the
        bins as necessary.

        note that unlike most adaptive smoothing algorithms, this one
        conserves flux.

syntax
       smoothie,y,outy,outye,ix,yerr=yerr,snrthr=snrthr,verbose=v

parameters
       y         [INPUT; required] array to be smoothed
       outy      [OUTPUT] smoothed array
       outye     [OUTPUT] errors on smoothed array
                 * WARNING: these do not include correlation errors,
                   and so do not represent a rigorous estimate.  i.e.,
                   even though correctly propagated for each bin, they
                   cannot be used as independent errors on the values in
                   adjacent bins.
       ix        [OUTPUT] bin indices -- tells which bins are grouped
                 * all bins which have the same value of IX are in the same group

keywords
       yerr      [INPUT] error on Y
                 * default is 1.+sqrt(Y+0.75)
                 * the default is set that way for consistency with
                   standard PINTofALE behavior, but SQRT(Y) is a better
                   option for this algorithm.
       snrthr    [INPUT] S/N threshold below which to smooth bins
                 * default is 3.0
        verbose  [INPUT] controls chatter

        _extra   [JUNK] here only to prevent crashing

example
        y=50*(mk_gauss(findgen(256),128,10,1)+0.01) & yr=long(0*y)
        for i=0,256-1 do yr[i]=randomu(seed,poisson=y[i])
        smoothie,yr,outy,outye,ix,yerr=1+sqrt(0.75+yr),snrthr=4.0
        plot,y & oplot,yr,psym=10,col=2 & oplot,outy,col=3
        ** notice the wings **
```

```
        y=randomu(seed,1000,poisson=1)
        y[400:499]=randomu(seed,100,poisson=5)
        smoothie,y,outy,outye,snrthr=5
        plot,y & oplot,outy,col=2,thick=3

these are all the smoothing tools in PINTofALE
        ALALOESS() makes a loess curve
        CLTSMOOTH() removes -ves from background subtracted spectra or light curves
        CONV_RMF convolves with specified RMF
        HAARTRAN() smoothes by filtering on Haar wavelet coefficients
        LINEREM() removes lines from spectrum by iterative S/N filtering
        NOISMOOTH() does boxcar accumulation a la Ebeling's asmooth
        REGROUP() accumulates from one end
        SCRMF does fast convolution using optimized RMF
        SMOOTHIE does peak-descent accumulation to build up S/N
        SPLAC computes a segmented piecewise linear approximations
        UNKINK() removes sharp kinks from a curve
        VARSMOOTH() does local smoothing at specified scales
        VOORSMOOTH() does fast local smoothing at specified scales

history
        VK/LL (Nov 04)
        cleaned up I/O (VK; Jan05)
        now returns a flat average if total(y)/sqrt(total(ysig^2))<SNRTHR
          (VK; Oct06)
        bugfix: unnecessarily crashing because of a variable name mismatch
          when too few counts to meet SNR threshold (VK; Oct09)
```

B.233 SMUDGE [/]

```
procedure       smudge
        returns labels for features at given locations

        the input file is expected to be in the following format:-
                LOCATION <sep> UNIT <sep> LABEL <sep> KEYWORDS
        where
          <sep> is a separator, usually a <tab>
          LOCATION is a number, usually the wavelength in [Ang]
          UNIT specifies the units for LOCATION, e.g., "Ang", "keV", etc.
          LABEL is a string label describing the feature at LOCATION
          KEYWORDS are a set of keywords that facilitate pinpoint extraction,
            usually the instrument name, the origin of the feature, etc.
        Note that:
          LOCATION is required, and everything else is optional
          Lines prefixed by "%", "#", ";", "/" are ignored
          the following UNITs are understood: Ang, eV, keV, MeV, Hz, MHz, GHz,
            wave-#, /cm, cm^-1, cm**-1, mu, micron, mm, cm, meter, km

syntax
        smudge,x,label,keys=keys,only=only,nisi=nisi,range=range,$
        infile=infile,sep=sep,prefix=prefix,/okeV,/help

parameters
        loc     [OUTPUT] float array of location of features
        label   [OUTPUT] string array of labels for features

keywords
        keys    [INPUT] string array of keywords.  if given, all features
                answering to these keywords are returned
        only    [INPUT] string array of keywords.  while KEYS does a logical
                OR, ONLY does a logical AND on the keywords
        nisi    [INPUT] return all >except< those features matching the
                specified keywords
                * KEYS is checked first, ONLY next, NISI last, and the
                  precedence is therefore in reverse order
        range   [INPUT] if set, returns only those LABEL(LOC)s that lie
                within RANGE
                * if 1-element vector or scalar, looks for exact match
                * WARNING: range check is carried out >after< conversion
                  to keV if OKEV (see below) is set, so RANGE must then be
                  given in units of [keV] (otherwise it should be in [Ang])
        infile  [INPUT] ascii file in which the features are listed
                * default: $SCARDIR/ardb/smudge.lst
```

```
        sep     [INPUT] separator used to delineate fields in INFILE
                * default is <tab>
        prefix  [INPUT] any line beginning with this is considered a comment
                * default: "*"
                * this is in addition to "%", "#", ";", "/"
                * beware: "<sp>" or "<tab>" won't work!
        okeV    [INPUT] if set, LOC are returned in units of keV
                * default is to return LOC in units of Ang
                * KEV is accepted as a synonym for historical reasons
        help    [INPUT] if set, prints out usage and quits
        ikeV    [INPUT; DEFUNCT] a synonym for OKEV, still kept
                (temporarily) only for backward compatibility

        _extra  [JUNK] here only to prevent crashing the program

example usage
        smudge,/help
        smudge,wvl,lbl & xr=[min(wvl),max(wvl)] & plot,[0],xr=xr,/xlog
                xyouts,wvl,randomu(seed,n_elements(wvl)),lbl,orient=90
        smudge,keys=['edge','gap'],only=['M']
        smudge,keys='gap',only=['S0','S1'],/okeV
        smudge,keys='edge gap',range=[1,6]
        smudge,keys='edge gap',nisi='edge',range=[1,6]
        smudge,range=0.69,/okev

history
        vinay kashyap (Nov98)
        added warning in case of bad SEP (VK; Dec98)
        changed "caldb" to "ardb" (VK; DecMM)
        added keyword IKEV in preparation to getting rid of KEV (VK; JanMMI)
        changed keyword IKEV to OKEV (VK; Jul02)
```

B.234 SOLAR_TRESP [/ssw]

```
function        solar_tresp
         Computes and returns temperature response for a selection of
        bandpasses from a variety of solar X-Ray and EUV diagnostic
        instruments. The explicitly supported instruments and bandpasses and their
        corresponding ouput units (at the sun) are as follows:

        TRACE 171,195,284 in [DN s-1 cm-2] per [cm-5 ln(K)-1]
        EIT   171,195,284 in [DN s-1 cm-2] per [cm-5 ln(K)-1]
        SXT   Al1, AlMg  in [DN s-1 cm-2] per [cm-5 ln(K)-1]
        RHESSI GD, in [Counts s-1] per [cm-5 ln(K)-1]
        CDS all GIS and NIS lines in [ergs s-1 cm-2] per [cm-5 ln(K)-1]

        The user may also specify an arbitrary instrument bandpass combination
        (one other than the explicitly supported ones listed above) with the
        limitation that the WRANGE keyword must be set. The units in this case
        would be [Counts s-1] per [cm-5 ln(K)-1] if and effective area is included and
        [ph s-1 cm+2] per [cm-5 ln(K)-1]  if it is not.

        A choice of atomic databases, abundances, and ion equilibria
        may be set via keywords LDBDIR,ABUND,and EQFILE respectively.
        the temperature response calculations are made like this:



syntax
        resp=solar_tresp(instrument,filter,date,tgrid=tgrid,ldbdir=ldbdir,$
        abund=abund,eqfile=eqfile)

parameters
        instrument  [INPUT;required] name of the relevant instrument
                    * can be 'TRACE','EIT','SXT','CDS','RHESSI'
                    * may also be set to a string descrption of
                      arbitrary instruments e.g. 'RESIK'. keyword
                      wrange must be set.
        bandpass    [INPUT;required] select the relevant bandpass
                    * can be '171','195','284' for 'TRACE' and 'EIT'
                    * can be 'Open','Al1','AlMg','Be119','AL12','MG3' for 'SXT'
                    * can be 'GD' for 'RHESSI'
                    * for 'CDS' must be a string specifiying an ionic
                      state (element abbreviation and roman
                      numeral) and a transition wavelength.
                      e.g.
                           'Mg VII 319.03'
```

```
                          'Ne VI 401.14'
                          'Fe IX 171.07'

                     CDS lines are treated as very narrow
                     bandpasses, so that possible contributions
                     from weak lines are accounted for.

                     The transition wavelength must fall into one of
                     the wavelength regions covered by GIS or NIS.

                     If the transition wavelength is designated WVL
                     and the detector resolution as RES, then the
                     bandpass wavelength range is determined
                     to be [WVL-3*RES,WVL+3*RES]. RES is
                     hardcoded as 0.18 Ang for NIS1, 0.32A for NIS12
                     and 0.21 for GIS. This range may be overridden by
                     using keyword WRANGE.

                  * can be set to and arbitrary string descrption
                    of bandpass if instrument is set to something other
                    than the explicitley supported instruments. e.g.
                    '1.123 - 1.293 AA' for instrument 'RESIK'
                    keyword WRANGE must be set.

keywords
       filter      [INPUT] Filter wheel selection for the EIT and
                   TRACE passbands
                   * default is 'clr' for EIT can also be 'al1', 'al2'
                   * default is 'AO' for TRACE can also be 'OO',
                     'OA' or 'AA'
                   * ignored if instrument selection is SXT or CDS
       date        [INPUT] time in any Yohkoh format or a Yohkoh
                     index structure.
                   * see e.g. ANYTIME2INTS() SolarSoft function for
                     standard Yohkoh formats
                   * SXT temperature responses time variable due to an enterance
                     filter failiure (Oct 1992). The value of this keyword will
                     be passed to GET_YO_DATES to determine appropriate correction.
                   * if unset no correction is made.
       wrange      [INPUT] a two elements floating point vector which
                     denotes a wavelength range. overrides the internally
                     determined bandpass range.
       tgrid       [INPUT] Log(T) grid on which the ouput response
                     is to be defined.
                   * default is PoA system variable !LOGT
```

```
                   * if !LOGT does not exist then tgrid=4.0+findgen(41)*0.1
       lstr        [I/O] line emissivity structure
       cstr        [I/O] continuum emissivity structure
       ldbdir      [INPUT] line emissivity database directory
                   * default is '$CHIANTI' can also be '$SPEX', '$APED'
                     or an explicit directory specification
                     e.g. '/data/foo/bar/spex/' or '../foo/bar/aped' etc.
       cdbdir      [INPUT] continuum emissivity database directory
                   * default is '$CONT'
       abref       [INPUT] string reference to the set of abundances to use
                   * default is 'feldman' which will call GETABUND() to
                   retrieve Feldman et al. (1992) abundances
                   * see GETABUND() documentation for other options.
       eqfile      [INPUT] string reference to the ion equilibria file to use
                   * default is 'ioneq/mazzotta_etal.ioneq'
                   * see local CHIANTI database for other options
       perpix      [INPUT] if set, responses are returned as
                   [DN s-1 pixel-1] per [cm-5 ln(K)-1].
                   * ignored for SOHO/CDS
       det         [INPUT] if set then responses are computed at detector
                     and not at the SUN
       rhessi_rmf  [INPUT] RHESSI spectrometer OGIP-compliant rmf(srm) fits file from wh
                     and effective area will be extracted via rd_ogip_rmf()
       rspstr      [OUTPUT] output structure containing all
                   ouputs and inputs as well as general info
                   concerning what went into response calculation.
                   * The structure is defined with these tags:
                     INSTR           STRING    instrument parameter
                     BANDPASS        STRING    bandpass parameter
                     FILTER          STRING    filter keyword
                     RESP            DOUBLE    computed response
                     TGRID           FLOAT     response Log(T) grid
                     EFFAR           DOUBLE    effective area
                     WVLAR           DOUBLE    effective area wavelength grid
                     LDBDIR          STRING    ldbdir keyword;
                     ABREF           STRING    abref keyword
                     EQFILE          STRING    eqfile keyword
                     ABUND           FLOAT     abund keyword
       nocont      [INPUT] if set, continuum will not be included in response
       zresp       [INPUT] if set,
       _extra       [INPUT] use to transmit defined keywords to called functions

restrictions
       * This function uses various SolarSoft routines
         and PINTofALE routines. It also employs files
```

```
         from Solar Soft databases. Note also that SolarSoft
         installations can vary in what missions/instruments
         are supported.
       * employs PINTofALE subroutines
         LICOSPEC
         LINEFLX
         GETABUND
         RD_LINE
       * employs SolarSoft subroutines
         SSW_INSTRUMENTS
         CONCAT_DIR
         RESTGEN
         GET_YO_DATES
         TRACE_EUV_RESP
         EIT_PARMS
       * If the DATE keyword is set, user cannot set the verbose keyword
         due to a conflict in GET_YO_DATES
history
      liwei lin (Aug 2004)
       BUGFIX (Jan 2005) had assumed that licospec output emissivities were in
        ergs cm-2 s-1. now /noph is fed through _extra to lineflx()
       BUGFIX (LL Sep 2005)  Thanks to Harry Warren who points out that upon reading the SXT
        effective area file, the FILTER entry already includes ENTRANCE, CCD,
        and MIRROR effective areas.
       BUGFIX (LL Sep 2005) the reported units, both in documentation and
        output structures were incorrect. Missing were terms 1d44 and cm^-5
       EDIT (LL Sep 2005) so that there is no hard reliance on PoA system
        variables. RHESSI default bandpass no longer 'G4'; user is asked
        if RHESSI SRM file not fed via RHESSI_RMF keyword.
       ADDED (LL Sep 2005) ZRESP keyword
       BUGFIX (LL Oct 2005) wvlar returned wmid rather than wvlar
       EDIT (LL Oct 2006) corrected reported UNITS
```

B.235 SPHTRIG_TRANSLB [/util]

```
function        sphtrig_translb
        transform the given {theta,phi} coordinates on a sphere that
        has been tilted and twisted (in that order) to {theta',phi'}
        on an untwisted and untilted pov reference frame, and
        returns them in a structure.

syntax
        tpstr=sphtrig_translb(theta,phi,tilt=tilt,twist=twist,$
        eps=eps,verbose=vv)

parameters
        theta   [INPUT; required] angular coordinate from Z-axis [deg]
        phi     [INPUT; required] azimuthal coordinate around Z-axis [deg]
                * THETA,PHI may be vectors or scalars
                * if vectors, assumed to be paired sets, with
                  values filled in with 0's if one falls short

keywords
        tilt    [INPUT; default=0] inclination angle of sphere
                relative to pov coordinate frame [deg]
        twist   [INPUT; default=0] rotation angle by which the
                vector going through the pole is rotated around
                the Z-axis of the pov reference frame [deg]
                * note that TWIST is given in the negative sense --
                  the angle by which the frame must be rotated to
                  return to 0.  the program takes the -ve of the
                  supplied TWIST.
        eps     [INPUT; default=1e-6] a small number
                * There are some singularities when THETA and PHI are zero.
                  This is added on to make sure at least something
                  close to the limiting value is returned in that case
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

example
        .run sphtrig_translb
        makes lines of constant latitude, constant longitude, and
        draws the figure used to derive the transformation used here.
        (the ultimate in self-documenting!)

subroutines
        example run requires PEASECOLR
```

```
history
        vinay kashyap (2013feb)
        added keyword EPS (2013mar)
        corrected sign error that was making PHI go clockwise instead of
          counter-clockwise (2015sep)
```

B.236 SPLAC [/util]

```
procedure       splac
        compute Segmented Piecewise Linear Approximation to Curve.
        generates piecewise linear approximations with fewest line
        segments within given tolerances.

about the name
        have you heard the one about the Stanford Linear Accelerator (SLAC)
        and what would happen after the Big One hits?  Obviously, it would
        have to be renamed SPLAC, the Stanford Piecewise Linear Accelerator..

syntax
        splac,x,y,ytol,u,v,w,/noband,/discon,verbose=verbose

parameters
        x       [INPUT; required] points at which data are defined
                * must be in ascending order
        y       [INPUT; required] values of data array, Y(X)
                * must match size of X
        ytol    [INPUT; required] tolerance of approximation: allowed
                >deviation< from the true curve
                * if scalar, single absolute-value tolerance for entire curve.
                * if vector, tolerance at each X -- size must match X
                * if -ve, ABS(YTOL) is taken to be the fractional factor
                  relative to the true value.
        u       [OUTPUT] a partition of the interval [X(0),X(N[X]-1)]
                with U(0)=X(0) and U(N[U]-1)=X(N[X]-1)
        v       [OUTPUT] ordinates, V(U)
                * the approximating segment is (u[i],v[i]) -> (u[i+1],v[i+1])
        w       [OUTPUT] ordinates if the approximation may be discontinuous
                * the approximating segment is (u[i],w[i]) -> (u[i+1],v[i+1])

keywords
        noband  [INPUT] indicates whether the approximation is to be
                restricted to the 'tolerance band' about the data
                (the 'tolerance band' is a piecewise linear band
                centered at the data whose width is determined by the
                tolerances at the data points.)
                * if set, indicates no band restriction
                * if not set, or set to 0, indicates apply this restriction
        discon  [INPUT] indicates whether or not the approximation must be
                continuous
                * if set, indicates continuity not required
                * if not set, or set to 0, indicates continuity is required
```

```
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

examples
        x=findgen(100) & y=sqrt(x)
        splac,x,y,0.1,u,v,w,/noband,verbose=10
        splac,x,y,0.3,u,v,w,/discon,verbose=10
        splac,x,y,-0.1,u,v,w & plot,x,y & oplot,u,v,psym=-1,col=100

subroutines
        NONE

restrictions
        Will not work with IDL versions < 5

these are all the smoothing tools in PINTofALE
        ALALOESS() makes a loess curve
        CLTSMOOTH() removes -ves from background subtracted spectra or light curves
        CONV_RMF convolves with specified RMF
        HAARTRAN() smoothes by filtering on Haar wavelet coefficients
        LINEREM() removes lines from spectrum by iterative S/N filtering
        NOISMOOTH() does boxcar accumulation a la Ebeling's asmooth
        REGROUP() accumulates from one end
        SCRMF does fast convolution using optimized RMF
        SMOOTHIE does peak-descent accumulation to build up S/N
        SPLAC computes a segmented piecewise linear approximations
        UNKINK() removes sharp kinks from a curve
        VARSMOOTH() does local smoothing at specified scales
        VOORSMOOTH() does fast local smoothing at specified scales

history
        From D.G. Wilson, 1976, ACM Transactions on Mathematical Software,2,388
        http://www.netlib.org/toms/510
          converted to IDL by Vinay Kashyap, who is very unhappy about all
          those GOTOs, but doesn't have the patience to rewrite from scratch
          (FebMM)
        bug correction for when YTOL was -ve; made into IDL5 (VK; JunMM)
        bug correction in plotting with /DISCON (VK; AprMMVII)
```

B.237 SQUISHEM [/]

```
function        squishem
        returns a new ID structure from the old one with all the
        multiple IDs (if any) squished into a single ID pointing
        to the one with the highest peak emissivity.  If the IDs
        include different elements, their emissivities are combined
        in the ratio of their abundances.

        a few more words about this combining:
        o the >fluxes< are combined without regard to abundance, i.e.,
          are added up directly.  this is because the fluxes are assumed
          to already include all info about DEMs, abundances, ion balance,
          and so on. besides, UPDATID splits a measured flux value simply
          on the basis of the ion-balanced emissivities.
        o the >emissivities< are combined by normalizing the weaker IDs to
          that of the strongest ID in the ratio of the abundances.  because
          the main ID itself only retains the strongest ID, the absolute
          abundance on the emissivity is not set, but relative abundances
          are fixed.

syntax
        sqID=squishem(idstr,abund=abund)

parameters
        idstr   [INPUT; required] an ID structure, e.g., output of LINEID

keywords
        abund   [INPUT] abundances relative to H (abund(0)=1)
                * abund(Z-1) contains the abundance for element Z
                * if array size is smaller than the largest present Z,
                  the last element is replicated to fill the gap
                * default: Anders & Grevesse (1989)
       mproxy  [OUTPUT] if set, will output 2 dimensional MPROXY string
               as prescribed by MIXIE()
        _extra  [JUNK] here only to avoid crashing the program

subroutines
        GETABUND
        ZION2SYMB
       INICON
history
        vinay kashyap (FebMM)
        bug correction with flux err (VK; MarMM)
        unknown IDs were crashing at abundances; added call to ZION2SYMB
```

```
   (VK; MMJul)
ADDED MPROXY keyword (Feb2004)
```

B.238 STAMPLE [/util]

```
procedure       stample
        brand existing plot.  The standard format is to put
        DATE (YYYY MON DD) TIME (HH:MM:SS) USER (name@host) PACKAGE
        at the lower-left of a plot.

        stample is a portmanteau (see Jabberwocky) of stamp and staple.

syntax
        stample,package,/nuthin,/noday,/notime,/nouser,/nopack,$
        stacol=stacol,stasiz=stasiz,stathk=stathk,/help

parameters
        package [INPUT] name of package
                * default is 'PINTofALE <version>'

keywords
        nuthin  [INPUT] if set, the same as setting all of
                NODAY,NOTIME,NOUSER,NOPACK
        noday   [INPUT] if set, does not include the day
        notime  [INPUT] if set, does not include the time
        nouser  [INPUT] if set, does not include the user name
        nopack  [INPUT] if set, does not include the package name
        stacol  [INPUT] index of color in color-table to use
                * default is !D.N_COLORS-10 or 1, whichever is greater
        stasiz  [INPUT] character size, defaults to 1.
        stathk  [INPUT] character thickness, defaults to 1.
        help    [INPUT] if set, prints usage and quits
        _extra  [JUNK] here only to avoid crashing the program

common
        cb_stample      {vname, vhost, vPoA}

restrictions
        requires subroutine SETSYSVAL

history
        vinay kashyap (DecMM)
        added common block STAMPLE (VK; JanMMI)
        added call to SETSYSVAL (VK; FebMMI)
        back-compatibility fix for SYSTIME (VK FebMMI/A.Maggio)
        bug corrected -- repeat use of "m" in call to CALDAT
          (VK Oct01/A.Maggio)
        changed name of common block from STAMPLE to CB_STAMPLE
```

`(VK Oct02)`

B.239 STARFLUX [/]

```
function        *flux
        returns total flux in the passband of interest at the specified
        temperatures OR for the given DEM

syntax
        flx=starflux(lstr,cstr,tlog=tlog,DEM=DEM,abund=abund,wrange=wrange,$
            NH=NH,effar=effar,wvlar=wvlar,dist=dist,radius=radius,/noph,$
            lflux=flux,cflux=cflux,fH2=fH2,He1=He1,HeII=HeII,/Fano)

a word on the units
        the line emissivities are in [ergs cm^3/s], and the continuum
        emissivities are also converted to this.  the [ergs] are converted
        to [ph] before output unless specified otherwise.
        thus, output will have units:
                [(ergs|ph) cm^3/s][EM][RADIUS^2][DIST^-2][AREA]
        e.g., if EM is [cm^-3] at source, and DIST is given, then output
        is [ph/cm^2/s], the flux at the telescope, which is fine.
        on the other hand, if DEM is [cm^-5] (n^e*V/(4*pi*D^2) at the
        telescope) and DIST are given, we'd get [ph/cm^4/s] which would
        be meaningless.  instead, RADIUS may be given to get [ph/s].
        *******************************************************************
        IT IS UP TO THE USER TO MAKE SURE THAT THE OUTPUT UNITS MAKE SENSE!
        *******************************************************************

parameters
        lstr    [INPUT; required] structure containing line emissivity data
                (see RD_LINE or RD_LIST)
                * BEWARE: ion balance is assumed to be included!
        cstr    [INPUT; required] structure containing continuum emissivities
                (see RD_CONT)

keywords
        tlog    [INPUT] EITHER [A] the temperatures at which to compute the
                output fluxes XOR [B] the temperatures at which the DEM is
                defined
                * if n(DEM) < 2, then [A]
                * if n(DEM) > 1 .AND. n(DEM) = n(TLOG), then [B]
        DEM     [INPUT] EITHER [A] volume emission measure [cm^-3 or cm^-5]
                XOR [B] differential emission measure [cm^-3/logK or cm^-5/logK]
                * if [B] then n(DEM) .MUST=. {n(TLOG) or n(LSTR.LOGT)}
                * default is 1e14
        abund   [INPUT] abundances
                * default: Grevesse et al. (1992)
```

```
        wrange  [INPUT] passband in which to compute flux
                * if not given, assumes larger of range(LSTR.WVL,CSTR.WVL)
        NH      [INPUT] ISM column density [cm^-2]
        effar   [INPUT] effective area [cm^2]
        wvlar   [INPUT] wavelengths at which effective area is defined [Ang]
                * n(EFFAR) .MUST=. n(WVLAR)
                * if not given, or illegal, output will be [ph/s/cm^2]
        dist    [INPUT] distance to star, in [pc]
                * BUT, if > 1e6, taken to be in [cm]
                * if given, divides computed flux by 4*pi*DIST^2 before output
                * BEWARE: no cosmological corrections are made!
        radius  [INPUT] radius of star, in [cm]
                * BUT, if < 1e6, taken to be in [RSol]
                * if given, multiplies flux by 4*pi*RADIUS^2 before output
                * BEWARE: no scale-height/volume effects included!
        noph    [INPUT] if set, does not convert [ergs] to [ph]
        lflux   [OUTPUT] flux due to lines only
        cflux   [OUTPUT] flux due to continuum only
                * LFLUX and CFLUX will >NOT< have DIST and RADIUS included!!
        _extra  [INPUT] pass defined keywords to
                ISMTAU [fH2,He1,HeII,Fano]
        ikeV    [JUNK] does nothing.  only to avoid passing it on to ISMTAU

restrictions
        requires subroutines ISMTAU, GETABUND, MK_DEM
        does not call LINEFLX or GRATFLX, so perhaps a wee bit faster,
          but on the other hand, cannot handle grating effective areas either.

history
        vinay kashyap (1999May)
        changed KEV to IKEV (VK; DecMM)
```

B.240 STR2ARR [/util]

```
function        str2arr
        converts a series of numbers written out as a string into an array.
        the output is an array of numbers of specified type

syntax
        arr=str2arr(str,delim,sep=sep,/i1,/i4,/r4,/r8,/squish,$
        array=array,delimit=delimit,/nomult)

parameters
        str     [INPUT; required] input string
                * if itself an array, is concatenated into one huge string
        delim   [INPUT] same as SEP, put here for consistency with SSWIDL
                version of program with same name.
                * if given, is used instead of SEP
                * if given, the output is returned as a string array
                  ignoring I1,I4,R4,R8
                * if not given and DELIMIT is not set, and I1,I4,R4,R8
                  are also not set, the output is an integer array

keywords
        sep     [INPUT] separator between entries in the string
                * if not given as a string, first checks "," and
                  if that doesn't seem to make any difference, then
                  checks "<sp>"
                * if explicitly set, checks only for SEP
                * if array, uses only the first element
        i1      [INPUT] if set, output is a BYTARR, default is I2
        i4      [INPUT] if set, output is a LONARR
        r4      [INPUT] if set, output is a FLTARR
        r8      [INPUT] if set, output is a DBLARR
                * precedence is R8 over R4 over I4 over I1 over I2
        squish  [INPUT] if set, "squishes" adjoining SEPs, e.g.,
                "1      2" is converted to "1 2"
        array   [OUTPUT] string array version of the output
        delimit [INPUT] delimiter, same as DELIM, and ignored if either
                DELIM or SEP are present
                * but if set, the output is a string array, and I1,I4,R4,R8
                  are ignored
        nomult  [INPUT] same as SQUISH, just for compatibility with SSWIDL
                version
        _extra  [JUNK] here only to prevent crashing the program

subroutines
```

```
        SYZE

a word
        yes, I know about READS.  it's limitations are that you still have
        to define the input array first, cannot use other values for the
        separator (e.g., ":"), and can't handle missing data (e.g., "1,2,")

history
        vinay kashyap (Nov98)
        added parameter DELIM and keywords ARRAY, DELIMIT, and NOMULT
          for the sake of consistency with SSWIDL version; also added
          call to SYZE() (VK; Jun02)
```

B.241 STR_2_ARR [/util]

```
function        str_2_arr
        converts a series of numbers written out as a string into an array.
        the output is an array of numbers of specified type

syntax
        arr=str_2_arr(str,delim,sep=sep,/i1,/i4,/r4,/r8,/squish,$
        array=array,delimit=delimit,/nomult)

parameters
        str     [INPUT; required] input string
                * if itself an array, is concatenated into one huge string
        delim   [INPUT] same as SEP, put here for consistency with SSWIDL
                version of program with same name.
                * if given, is used instead of SEP
                * if given, the output is returned as a string array
                  ignoring I1,I4,R4,R8
                * if not given and DELIMIT is not set, and I1,I4,R4,R8
                  are also not set, the output is an integer array

keywords
        sep     [INPUT] separator between entries in the string
                * if not given as a string, first checks "," and
                  if that doesn't seem to make any difference, then
                  checks "<sp>"
                * if explicitly set, checks only for SEP
                * if array, uses only the first element
        i1      [INPUT] if set, output is a BYTARR, default is I2
        i4      [INPUT] if set, output is a LONARR
        r4      [INPUT] if set, output is a FLTARR
        r8      [INPUT] if set, output is a DBLARR
                * precedence is R8 over R4 over I4 over I1 over I2
        squish  [INPUT] if set, "squishes" adjoining SEPs, e.g.,
                "1      2" is converted to "1 2"
        array   [OUTPUT] string array version of the output
        delimit [INPUT] delimiter, same as DELIM, and ignored if either
                DELIM or SEP are present
                * but if set, the output is a string array, and I1,I4,R4,R8
                  are ignored
        nomult  [INPUT] same as SQUISH, just for compatibility with SSWIDL
                version
        _extra  [JUNK] here only to prevent crashing the program

subroutines
```

```
        SYZE

a word
        yes, I know about READS.  its limitations are that you still have
        to define the input array first, cannot use other values for the
        separator (e.g., ":"), and can't handle missing data (e.g., "1,2,")

history
        vinay kashyap (Nov98)
        added parameter DELIM and keywords ARRAY, DELIMIT, and NOMULT
          for the sake of consistency with SSWIDL version; also added
          call to SYZE() (VK; Jun02)
        changed name from STR2ARR to STR_2_ARR to avoid conflict with
          SSW routine because of compilation issues with SSW (VK; Apr05)
```

B.242 SUTHERLAND_CH [/external]

PROJECT: CHIANTI

CHIANTI is an Atomic Database Package for Spectroscopic Diagnostics of Astrophysical Plasmas. It is a collaborative project involving the Naval Research Laboratory (USA), the University of Florence (Italy), the University of Cambridge and the Rutherford Appleton Laboratory (UK).

NAME:

SUTHERLAND_CH

PURPOSE:

Calculate the free-free continuum from an hot, low density plasma

Uses the free-free gaunt factor calculations of Sutherland, 1998, MNRAS, 300, 321.

Note that Sutherland's Eq.(15) has units of erg/cm^3/s. Comparing with Rybicki & Lightman's Eq.5.14(a) (in their book 'Radiative Processes in Astrophysics'), suggests that Sutherland's units should be erg/cm^3/s/sr/Hz. We are assuming the latter to be correct in this routine.

When using the DEM_INT optional input, FREEFREE expects the differential emission measure to have been derived from a product of N_e*N_H rather than N_e*N_e. This can be important when dealing with a regime (typically T < 10^4.5) where H and He are not fully ionised.

To calculate the continuum spectrum the user should specify whether the plasma structure is specified through a differential emission measure (DEM) curve or through emission measure (EM) values. For the former, the DEM values are specified through the DEM_INT keyword and for the latter the EM values are specified through the EM_INT keyword. If neither are specified EM_INT values of unity are assumed.

If DEM_INT values are specifed they must be specified on a temperature scale that has constant intervals in log10(T). EM_INT values do not have this restriction.

Note that EM_INT must be used for isothermal plasmas.

```
CALLING SEQUENCE:

      SUTHERLAND_CH,temperature, wavelength, intensity


INPUTS:

       T          Temperature in degrees Kelvin, can be a 1 D
                 array. If the keyword /DT is set, then the
                 temperatures must be specified with a fixed spacing
                 in logT. E.g., TEMP=10.^[4.0,4.1,4.2].

      WVL        Wavelength in Angstroms. If /keV is set, then WVL is
                 assumed to contain energies in keV units.


OPTIONAL INPUTS:

       None

KEYWORD PARAMETERS:

       NO_SETUP:  If the procedure setup_elements has already been called
                 then the keyword /no_setup should be set to avoid
                 repeating this step.

      MIN_ABUND: If set, calculates the continuum only from those
                 elements which have an abundance greater than min_abund.
                 This can speed up the calculations. ;

      DEM_INT   This should be the same size as TEMP and contain
                differential emission measure values. Specifying
                DEM_INT places a restriction on TEMP (see above). Note
                that DEM_INT values are multiplied by the factor dT
                within the routine, whereas EM_INT values are not.

      EM_INT    This should be the same size as TEMP and contain
                emission measure values. Note
                that DEM_INT values are multiplied by the factor dT
                within the routine, whereas EM_INT values are not. If
                neither EM_INT or DEM_INT are specified, then a EM_INT
                vector of same size as TEMP with values of unity is
                assumed.
```

```
      PHOTONS    The output spectrum is given in photon units rather
                 than ergs.

      SUMT       The default is to output the intensity array as an array
                 of size (nwvl x nT). Setting this keyword performs a sum
                 over the temperatures to yield a vector of same size as
                 the input wavelengths, thus producing the complete
                 free-free spectrum.

      KEV        If set, then WVL is assumed to contain energies in keV
                 units rather than wavelengths in Angstroms.

OUTPUTS:

       RAD        Free-free continuum intensity in units
                 10^-40 erg cm^3 s^-1 str^-1 angstrom^-1 per unit
                 emission measure [ integral (N_e N_H dh) in cm^-5 ].
                 If T is given as a 1-D array, then RAD will be output
                 as a 2-D array, with one element for each temperature
                 and wavelength (but also see SUMT).

                 If the keyword /keV is set, then the units of INT will be
                 10^-40 erg cm^3/s/sr/keV

PROGRAMMING NOTES

    The gaunt factors from Sutherland (MNRAS 300, 321, 1998) are a
    function of uu and gg (see his Eq. 14). uu is a function of both
    wavelength and T, while gg is a function of T only.

    The gaunt factor (gff) is tabulated for values of uu and gg at
    fixed intervals in log(uu) and log(gg). The log(uu) values go from
    -4 to 4 in 0.2 steps; the log(gg) values go from -4 to 4 in 0.1 steps.

    A particular (input) temperature and wavelength give rise to values
    uu0 and gg0, the logs of which lie between -4 and 4. To derive the
    corresponding gff0 value, I use the IDL routine BILINEAR.

    BILINEAR requires, not uu and gg values as input, but indices.
    E.g., the indices corresponding to the tabulated values of uu are
    0 (=-4.0), 1 (=-3.8), 2 (=-3.6), etc. Thus, if log(uu0)=-3.76, then
    i_uu0=1.20 is the index of uu0.

    In order to make significant time-savings, I give BILINEAR all of
    the wavelengths and temperatures in the same call for a particular
```

```
    ion. To do this, I make my i_uu and i_gg values 2-D arrays of size
    (nwvl x nT), and BILINEAR then produces a (nwvl x nT) array
    containing the gff values.

    A problem occurred if nT=1, as BILINEAR will turn the input uu and
    gg vectors into 2-D arrays of size (nwvl x nwvl). If there are a
    large number of wavelengths, this uses a lot of memory. To solve
    this I make a 2 element temperature vector whose values are
    identical, and then change this back to a 1 element vector after
    BILINEAR has been called. See the parts of the code where I use
    'tst1'.


COMMON BLOCKS:

       common elements,abund,abund_ref,ioneq,ioneq_t,ioneq_ref

CALLS

      READ_IP, READ_GFFGU, SETUP_ELEMENTS

EXAMPLES:

      IDL> freefree,1.e+6,wvl,int
      IDL> freefree,1.e+6,wvl,int,min_abund=3.e-5
      IDL> freefree,1.e+6,wvl,int,/no_setup,min_abund=1.e-6

      IDL> wvl=findgen(5001)/10. + 50.
      IDL> temp=10.^(findgen(41)/10. +4.)
      IDL> freefree,temp,wvl,int


MODIFICATION HISTORY:
       Written by:     Ken Dere
       March 1999:     Version 2.0
      September 1999:  Version 3.0

      Ver.3.1, 11-Aug-00, Peter Young
          Improved call to bilinear, allowing routine to solve for
          all temperatures in one go. This makes the routine quicker,
          and also lowers the memory usage of the routine when dealing
          with many wavelengths.

      Ver.3.2, 16-Aug-00, Peter Young
          Corrected expression for 'gg', replacing ip with z^2.
```

```
Ver.3.3, 16-Oct-00, Peter Young
    Now deals with dem_int correctly

Ver.3.4, 10-Oct-01, Ken Dere
    Corrected for labelling errors in Sutherland's gffgu.dat file
    No longer reads ionization potential file

Ver.3.5, 5-Dec-01, Peter Young
    Corrected expression for gamma^2
    Renamed routine sutherland.pro
    Restructured code to make it run quicker.

Ver.3.6, 22-May-01, Peter Young
    Re-instated the MIN_ABUND optional input.
    Changed ioneq_t to ioneq_logt (GDZ).

Ver.3.7, 18-Aug-03, Peter Young
    Activated /PHOTONS keyword

Ver.3.8, 5-Nov-03, Peter Young
    Corrected bug found by Jim McTiernan when multiple temperatures
    were input. The quantity 'newfactor' was not being calculated
    correctly due to indexing problems.

Ver. 4, 9-Mar-2006, Peter Young
    Added /keV keyword.

Ver.5, 10-Sep-2009, Peter Young
     Routine now checks that temperatures are evenly spaced
     when DEM_INT is specified. Also 0.1 dex spacing is no
     longer assumed.

Ver.6, 6-Oct-2009, Peter Young
     Introduced EM_INT keyword for specifying emission measure
     values for isothermal plasmas.

Ver.Poa, 10-Aug-2010, Vinay Kashyap
     a la 18-Nov-2005, LiWei Lin
    Commented out common block
    Added keywords,abund, ioneq, ieq_logt, _extra
    Changed ioneq_logt to ieq_logt
    Renamed routine sutherland_ch
```

B.243 SYMB2ZION [/util]

```
procedure       symb2zion
                convert spectroscopic symbol to atomic number and ionic state

syntax
        symb2zion,symbol,z,ion

parameters
        symbol  [INPUT; required] string containing element designation
                (e.g., "he_2", "He II", "HeII", "he 2", "he", "He", whatever)
        z       [OUTPUT] atomic number
        ion     [OUTPUT] ionic state (0=undefined, 1=neutral, 2=singly
                ionized, etc.)

keywords
        verbose [INPUT] higher the number, greater the chatter
        _extra  [JUNK] here only to prevent crashing the program

restrictions
        * cannot handle the "+/-" type notation (e.g., O^+5)
        * currently coded only up to Es

requires
        INICON
        LAT2ARAB

history
        written by vinay kashyap (Nov96), based on CHIANTI's convertname.pro
        modified screen dump to also show input (VK; Dec97)
        added call to INITSTUFF, include elements up to Es (VK; Aug98)
        corrected bug that was returning garbage for single-letter elements
          (VK; Oct98)
        was returning 0 for ions for, e.g., "O   VII"; corrected (VK; May99)
        changed call to INITSTUFF to INICON (VK; 99May)
        added keyword VERBOSE (VK; MarMM)
        added various trans-Uranium symbols (VK; AugMM)
        now allow tab separation (VK; Mar03)
```

B.244 SYZE [/util]

```
function        syze
        returns a vector that contains the size and type information a la
        IDL information routine SIZE.  The diffrence is that this one
        operates on strings only

        output is always of longward type.
        the first element contains the number of elements in C,
        the next elements contain the string-length of each element, one
          element per element of C,
        the last but one element contains the type code:
                0 : undefined,
                1 : I*1
                2 : I*2
                3 : I*4
                4 : R*4
                5 : R*8
                -1 : expression
                -2 : equation
                7 : everything else
        the last element contains the total number of characters in C

syntax
        sc=syze(c,oc=oc)

parameters
        c       [INPUT] string expression to decode

keywords
        oc      [OUTPUT] converted output, where applicable
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (Dec98)
        now catches sexagesimal (VK; Aug03)
```

B.245 TAG_EXIST [/external]

```
Project     : SOHO - CDS

Name        : TAG_EXIST()

Purpose     : To test whether a tag name exists in a structure.

Explanation : Routine obtains a list of tagnames and tests whether the
              requested one exists or not. The search is recursive so
              if any tag names in the structure are themselves structures
              the search drops down to that level.  (However, see the keyword
               TOP_LEVEL).

Use         : IDL>  status = tag_exist(str, tag)

Inputs      : str  -  structure variable to search
              tag  -  tag name to search for

Opt. Inputs : None

Outputs     : Function returns 1 if tag name exists or 0 if it does not.

Opt. Outputs: None

Keywords    : INDEX      = Index of matching tag

               TOP_LEVEL = If set, then only the top level of the structure is
                           searched.
              RECURSE  = set to recurse on nested structures

Category    : Util, structure

Written     : C D Pike, RAL, 18-May-94

Modified    : Version 1.1, D Zarro, ARC/GSFC, 27-Jan-95
              Passed out index of matching tag
               Version 2, William Thompson, GSFC, 6 March 1996
                       Added keyword TOP_LEVEL
              Version 2.1, Zarro, GSFC, 1 August 1996
                      Added call to help
              Version 3, Zarro, EIT/GSFC, 3 June 2000
                      added check for input array structure
              Version 4, Zarro, EIT/GSFC, 23 Aug 2000
                      removed calls to DATATYPE
```

```
Version 5, Zarro, EIT/GSFC, 29 Sept 2000
        added /quiet
Version 6, Zarro (EER/GSC), 22 Dec 2002
        made recursion NOT the default
```

B.246 TEKHI [/util]

```
function chitex
       Converts spectroscopic terms and configurations to LaTeX
       form, e.g.:
        '1s.2p 3P2.0'      to '$1s2p \; ^3P_{2}$'
        '1s.2p 3P2.5'      to '$1s2p \; ^3P_{5/2}$'
        '(2s2.2p5) 2P_3'   to '$(2s^22p^5) \; ^2P_{3}$'
        '(2s2.2p5) 2P_3/2' to '$(2s^22p^5) \; ^2P_{3/2}$'

syntax
        outstring = tekhi(instring)

parameters
       instring [INPUT; required] single string that describes the
                electron configurations and/or level designations to be
                converted to LaTex form.

keywords
       nodollar [INPUT] if set then output will not be wrapped with '$'s

        _extra   [JUNK] here only to prevent crashing the program

subroutines
          STREGEX
          STRJOIN
          STRMID

history
       LiWei Lin/Jeremy Drake (Nov04)
        added keyword _EXTRA and cleaned up usage (VK; Jul05)
```

B.247 TEKKU [/util]

```
procedure       tekku
        write out a LaTeX file summary of the ID structure

syntax
        tekku,id_str,fx,fxerr,texfil=texfil,fxunit=fxunit,/trance,/asku,$
        fxfmt=fxfmt,fxefmt=fxefmt,/kamentu

parameters
        id_str  [INPUT; required] ID structure from LINEID
        fx      [INPUT] measured fluxes
                * if size does not match number of ID components of the
                  number of IDs, the internally stored values are used
                * if FX is set to 0, then the output will not contain
                  any information about fluxes.
        fxerr   [INPUT] errors on FX
                * if size does not match FX, first element is assumed
                  to represent a constant >fractional< error on FX
                * default is 0.01
                * if FXERR is explicitly set to 0, then the output will
                  not report the errors.

keywords
        texfil  [INPUT] name of file to dump the table to
                * the ".tex" extension is not needed
                * if not given, dumps output to screen
                * if ASKU is set, the default extension is ".txt"
        fxunit  [INPUT] string representing the units that the fluxes
                are in, e.g., "ph s$^{-1}$"
        trance  [INPUT] if set, adds a column to hold transition data
                (e-configuration and level designation)
       trantex [INPUT] if set, e-configuration and level designation
               are converted to LaTex format so that appropriate
               super-scripting and sub-scripting is included. This is set
               by default.
                * explicitly set to zero to turn this off.
        asku    [INPUT] if set, writes out a plain text ascii file w/o all
                the TeX commands
        fxfmt   [INPUT] FORTRAN-style formatting syntax for writing out
                fluxes, e.g., "f10.4", "e14.4", etc.
                * default is "f7.2"
        fxefmt  [INPUT] as FXFMT, for flux errors
                * default is FXFMT
        kamentu [INPUT] set this keyword to include the scracth pad notes
```

```
                in the table output.  if not set, the comments won't be
                included.
        lineu   [INPUT] if set, the input structure is assumed to be a
                line emissivity structure of the sort generated by RD_LINE()
                * in this case, the lambda_obs column is not printed out,
                  and the lambda_pred column is simply labeled as lambda
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (Dec98)
        added support for scribbles (VK; Mar99)
        changed call to INITSTUFF to INICON (VK; 99May)
        allowed output minus flux+-fluxerr column; also transition info
          column (VK; 99Nov)
        added keyword ASKU (VK; MarMM)
        added keywords FXFMT and FXEFMT (VK; JunMM)
        corrected: what if FXFMT/FXEFMT not set (VK NovMM; viz. A.Maggio)
        added keyword KAMENTU; changed transitions font to "\small";
          now reads in flux and fluxerr from ID_STR; Erica asks that
          wvls, and flux+error alignments be set to "r" (VK; Apr02)
        now puts "..." instead of spaces for duplicates (VK; Oct03)
       added support for TEKHI() via keyword TRANTEX (LL;Nov04)
        added keyword LINEU, call to DUMMYID(), changed \nl to \\ (VK; Feb06)
        changed \documentstyle[apjpt4]{article} to \documentclass[10pt]{article}
          (VK; Mar07)

etymology
        the name comes from kannadization of the "process of TeXing";
        Kannada, mind you, is a language of southern India, and has
        absolutely nothing to do with Quebec and surrounding regions.
```

B.248 TI_AND [/timing]

```
procedure       ti_and
        returns the intersection of two sets of time intervals as
        defined by the start and stop times

syntax
        ti_and,tstr1,tstp1,tstr2,tstp2,tstart,tstop

parameters
        tstr1   [INPUT; required] start times for first set
        tstp1   [INPUT; required] stop times for first set
        tstr2   [INPUT; required] start times for second set
        tstp2   [INPUT; required] stop times for second set
        tstart  [OUTPUT] start times of resultanat intervals
        tstop   [OUTPUT] stop times of resultanat intervals

keywords
        _extra  [JUNK] here only to prevent crashing the program

restrictions
        inputs are assumed to be sorted in ascending order

related
        TI_CLEAN        - cleans up intervals
        TI_OR           - merges intervals
        TI_WRITE        - writes intervals to file
        TI_COVER()      - computes coverage in given bin
        TI_FILTER()     - filter an array by GTI

history
        vinay kashyap (May98)
        added keyword _EXTRA (VK; Nov98)
```

B.249 TI_CLEAN [/timing]

```
procedure       ti_clean
        given an arbitrary set of starting and stopping times of intervals,
        first organizes them in ascending order and then merges the various
        intervals that are contiguous.

syntax
        ti_clean,tstart,tstop,leap=leap

parameters
        tstart  [I/O; required] starting times of intervals
        tstop   [I/O; required] ending times of intervals

keywords
        leap    [INPUT; default=1.] if the difference between a stopping
                time and the *next* starting time of sequential intervals
                is < LEAP, merge the two intervals into one.
        _extra  [JUNK] here only to prevent crashing the program

related
        TI_OR           - merges intervals
        TI_AND          - intersection of intervals
        TI_WRITE        - writes intervals to file
        TI_COVER()      - computes coverage in given bin
        TI_FILTER()     - filter an array by GTI

history
        vinay kashyap (May98)
        added keyword _EXTRA (VK; Nov98)
        now also handles overlapping bad times (VK; Mar99)
```

B.250 TI_COVER [/timing]

```
function        ti_cover
        compute and return the good time coverage fraction for each
        bin of a time grid, such as those used to generate lightcurves

syntax
        tcover=ti_cover(tstart,tstop,tgrid,tmin=tmin,tmax=tmax,tbin=tbin,$
        verbose=verbose,/slowok)

parameters
        tstart  [INPUT; required] array of start times of GTI's
        tstop   [INPUT; required] array of stop times of GTI's
        tgrid   [I/O] the bin boundaries of the grid for which the
                coverage must be calculated
                * if undefined, or has less then 2 elements, will
                  be calculated internally using the keywords
                  TMIN, TMAX, and TBIN, and will be overwritten on output
                * if legal, TMIN, TMAX, and TBIN will be ignored and
                  the values will not have been changed on output

keywords
        tmin    [INPUT; default=min(TSTART)] time binning grid minimum
        tmax    [INPUT; default=max(TSTOP)] time binning grid maximum
        tbin    [INPUT; default=TMAX-TMIN] the binsize of a regular grid
                for binning
        verbose [INPUT] controls chatter
        _extra  [INPUT ONLY] pass defined keywords to subroutines
                REBINW: SLOWOK (skip call to FINDEX and use INTERPOL instead)

related
        TI_CLEAN        - cleans up intervals
        TI_OR           - merges intervals
        TI_AND          - intersection of intervals
        TI_WRITE        - writes intervals to file
        TI_FILTER()     - filter an array by GTI

subroutines
        REBINW
        FINDEX

history
        vinay kashyap (MMVI.IX)
```

B.251 TI_DEGTI [/timing]

```
function        ti_degti
        given the event times and GTI start and stop times, removes the
        gaps from the input and returns a set of points that run continuously.
        useful for when there are large gaps in the data or large numbers of
        breaks, and the source does not have any periodicities

syntax
        tx=ti_degti(tarr,tstart,tstop,tdelt,igti=igti,startt=startt,stopt=stopt,$
        verbose=verbose, leap=leap)

parameters
        tarr    [INPUT; required] input time array
        tstart  [INPUT; required] array of GTI start times
        tstop   [INPUT; required] array of GTI stop times
                * TSTART and TSTOP must match in size
        tdelt   [OUTPUT] add this to the output to get back the input

keywords
        igti    [OUTPUT] index array that says which interval each element of TARR is in
                * NOTE: this does not necessarily match TSTART and TSTOP indices,
                  because this program first cleans them up, and could shrink the
                  number of distinct intervals.
        startt  [OUTPUT] cleaned TSTART
        stopt   [OUTPUT] cleaned TSTOP
        verbose [INPUT] controls chatter
        _extra  [JUNK] pass defined keywords to subroutines
                TI_CLEAN: LEAP

subroutines
        TI_CLEAN

example
        .run ti_degti

history
        Vinay Kashyap (2015sep)
```

B.252 TI_FILTER [/timing]

```
function        ti_filter
        returns an array of locations where the input list of times
        fall within a given time interval defined by the start and
        stop times

syntax
        it=ti_filter(time,tstart,tstop)

parameters
        time    [INPUT; required] arrival times of photons, or bin-values
        tstart  [INPUT; required] start times of intervals
        tstop   [INPUT; required] stop times of intervals

keywords
        count   [OUTPUT] number of elements in output array
        _extra  [JUNK] here only to prevent crashing the program

related
        TI_CLEAN        - cleans up intervals
        TI_OR           - merges intervals
        TI_AND          - intersection of intervals
        TI_WRITE        - writes intervals to file
        TI_COVER()      - computes coverage in given bin

history
        vinay kashyap (May98)
        added keyword _EXTRA (VK; Nov98)
```

B.253 TI_OR [/timing]

```
procedure       ti_or
        returns the union of two sets of time intervals as
        defined by the start and stop times

syntax
        ti_or,tstr1,tstp1,tstr2,tstp2,tstart,tstop

parameters
        tstr1   [INPUT; required] start times for first set
        tstp1   [INPUT; required] stop times for first set
        tstr2   [INPUT; required] start times for second set
        tstp2   [INPUT; required] stop times for second set
        tstart  [OUTPUT] start times of resultanat intervals
        tstop   [OUTPUT] stop times of resultanat intervals

keywords
        _extra  [JUNK] here only to prevent crashing the program

restrictions
        inputs are assumed to be sorted in ascending order
        requires subroutine TI_CLEAN

related
        TI_CLEAN        - cleans up intervals
        TI_AND          - intersection of intervals
        TI_WRITE        - writes intervals to file
        TI_COVER()      - computes coverage in given bin
        TI_FILTER()     - filter an array by GTI

history
        vinay kashyap (May98)
        added keyword _EXTRA (VK; Nov98)
```

B.254 TI_WRITE [/timing]

```
procedure       ti_write
        writes out the start and stop times of a set of time intervals

syntax
        ti_write,tstart,tstop,tifile,fitsref=fitsref,exten=exten,$
        form=form,fitshdr=fitshdr,page=page

parameters
        tstart  [INPUT; required] start times
        tstop   [INPUT; required] stop times
        tifile  [INPUT] name of file to write to (default: STDOUT)
                * if name ends in .fit, .fits, .ft, or .mt then
                  write out a FITS file with a bintable extension
                  else an ascii file
                * if filename contains 'none' or 'stdout', will print to STDOUT

keywords
        fitsref [INPUT] name of existing FITS file from which to
                steal the header
        exten   [INPUT] extension of FITSREF from which steal header
        fitshdr [INPUT] explicitly specify the header.
                * if FITSHDR is set, FITSREF and EXTEN will be ignored!
        form    [INPUT] format in case of ASCII/STDOUT output
                * default: '(d16.4,1x,d16.4)'
                * ignored for FITS output
        page    [INPUT] number of lines to display on the screen at a time
                * default: 22
                * ignored if not writing to STDOUT
        _extra  [JUNK] here only to prevent crashing the program

related
        TI_CLEAN        - cleans up intervals
        TI_OR           - merges intervals
        TI_AND          - intersection of intervals
        TI_COVER()      - computes coverage in given bin
        TI_FILTER()     - filter an array by GTI

history
        vinay kashyap (May98)
        added keyword _EXTRA (VK; Nov98)
        bug correction with setting TIFILE='STDOUT' (VK; Sep06)
        slightly more verbose prompt for PAGE (VK; Aug15)
```

B.255 TIEUP1 [/scrypt]

```
script  tieup1

given an ID structure (IDSTR)
        -- sets the line width of the first component to LSFWDT,
        -- ties all the other widths to the first one,
        -- freezes all but the first width parameter,
        -- ties all line locations to that of the first one, but
           with a leeway defined by LINERR, and
        -- if variable POSITIVF is set, then constrains all fluxes
           to be +ve

outputs are
        TIES    a string array containing all the ties
        THAW    integer array signifying frozen(0) and thawed(1) parameters
        POS     a float array containing the initial positions
        WDT     a float array containing the initial widths
        HGT     a float array containing the initial heights

then, if
        SPECT(LAMBDA) is well defined, goes ahead and calls FITLINES
        and makes a new copy of IDSTR -- IDSTR2 -- with the new fluxes.

the fit parameters are in
        FPAR    a structure containing everything of interest
        POS,WDT,FLX     best-fit values
        [PWF]ERR[UL]    upper and lower deviations on best-fit values
        [PWF]ERRC       delta-chisq threshold for which deviations are computed

vinay kashyap (MarMM)
        added EPITHET to call (VK; MMaug)
        now LINERR can be 0 (VK; MVIMaug)
```

B.256 TIEUP2 [/scrypt]

```
script  tieup2

differs from TIEUP1 in that the line widths are not all identical
to the first one, but are allowed some wiggle room.

given an ID structure (IDSTR)
        -- sets the line width of the first component to LSFWDT,
        -- ties all other line widths to be WDT[0] +- WDTERR
        -- ties all line locations to that of the first one, but
           with a leeway defined by LINERR
        -- if variable POSITIVF is set, then constrains all fluxes
           to be +ve

outputs are
        TIES    a string array containing all the ties
        THAW    integer array signifying frozen(0) and thawed(1) parameters
        POS     a float array containing the initial positions
        WDT     a float array containing the initial widths
        HGT     a float array containing the initial heights

then, if
        SPECT(LAMBDA) is well defined, goes ahead and calls FITLINES
        and makes a new copy of IDSTR -- IDSTR2 -- with the new fluxes.

the fit parameters are in
        FPAR    a structure containing everything of interest
        POS,WDT,FLX     best-fit values
        [PWF]ERR[UL]    upper and lower deviations on best-fit values
        [PWF]ERRC       delta-chisq threshold for which deviations are computed

vinay kashyap (MarMM)
        added EPITHET to call (VK; MMaug)
```

B.257 TIMEVARKS [/timing]

```
function        timevarks
        computes the probability that the given set of photon arrival
        times are obtainable as a random instance from a constant source.

        smaller values imply higher significances of variability.

syntax
        vsigni=timevarks(times,dobs,pobs,tstart=tstart,tstop=tstop,$
        nsim=nsim,verbose=verbose)

parameters
        times   [INPUT; required] photon arrival times
                * WARNING: this routine does not handle dead-time
                  intervals, primbsching, or even GTIs
        dobs    [OUTPUT] the deviation found in the cdfs between
                data and the flat lightcurve model
        pobs    [OUTPUT] significance of DOBS, as calculated for
                the one-sample Kolmogorov-Smirnoff test

keywords
        tstart  [INPUT] beginning times of each segment of the
                good time intervals
        tstop   [INPUT] ending times of each segment of the GTIs
                * array sizes of TSTART and TSTOP must match, or
                  else they will be ignored
                * if in any case TSTOP < TSTART, they will be flipped
        nsim    [INPUT; default=1000] number of simulations
                to run to determine the significance
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

subroutines
        FLAT_LC_CDF (included)
        KSONE (from IDLAstro)
        PROB_KS (from IDLAstro)
        TI_CLEAN (in $PINTofALE/pro/timing/ti_clean.pro)

restrictions
        keyword values are passed on to FLAT_LC_CDF via KSONE, so
        make sure that KSONE allows that (post Apr 2005)

description
        first does an one-sample K-S test on the data, compared to
```

```
        a flat lightcurve model, to determine the deviation and its
        significance.  the observed deviation is then calibrated
        with one-sample K-S tests using randomly generated events
        from a flat lightcurve (also compared to a flat lightcurve
        model) and comparing the distribution of the deviations thus
        found with the observed deviation.

history
        vinay kashyap (MMVI.VIII)
        added keywords TSTART and TSTOP (VK; MMVI.IX)
        bug: GTI filtering broken, tzero offsets incorrect, and
          FLAT_LC_CDF was changing TMIN,TMAX (VK; MMVII.IV)
```

B.258 TIMEVARVK [/timing]

```
function        timevarvk
        computes the probability that the given set of photon arrival
        times are obtainable as a random instance from a constant source.

        smaller values imply higher significances of variability.

        uses a form of Kolmogorov-Smirnoff test, as in TIMEVARKS(), but
        differs in an important feature: the deviations between the data
        cdf and the model cdf are _averaged_ over a predefined number of
        consecutive photons.  the reasoning behind this is that the K-S
        statistic is a point estimate, whereas a real variability feature
        always shows some width, i.e., the intensities in the adjacent bins
        in a lightcurve are correlated.  the averaging of the distance
        differences will suppress the random Poisson deviations but not
        the actual signal.

syntax
        vsigni=timevarvk(times,dobs,pobs,tstart=tstart,tstop=tstop,$
        nsim=nsim,evtavg=evtavg,verbose=verbose)

parameters
        times   [INPUT; required] photon arrival times
                * WARNING: this routine does not handle dead-time
                  intervals, primbsching
        dobs    [OUTPUT] the deviation found in the cdfs between
                data and the flat lightcurve model
        pobs    [OUTPUT] significance of DOBS, as calculated for
                the one-sample Kolmogorov-Smirnoff test

keywords
        tstart  [INPUT] beginning times of each segment of the
                good time intervals
        tstop   [INPUT] ending times of each segment of the GTIs
                * array sizes of TSTART and TSTOP must match, or
                  else they will be ignored
                * if in any case TSTOP < TSTART, they will be flipped
        nsim    [INPUT; default=1000] number of simulations
                to run to determine the significance
        evtavg  [INPUT] the number of events on either side of the
                current event over which the distance deviations
                must be averaged over; i.e., a boxcar smoothing scale
                of 2*EVTAVG+1 is used
                * if not given, assumed to be 1
```

```
                * set explicitly to 0 to avoid doing the averaging;
                  this should produce results identical to that of
                  TIMEVARKS()
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

subroutines
        FLAT_LC_CDF (included)
        PROB_KS (from IDLAstro)
        TI_CLEAN (in $PINTofALE/pro/timing/ti_clean.pro)

description
        first does an one-sample K-S test on the data, compared to
        a flat lightcurve model, to determine the deviation and its
        significance.  the observed deviation is then calibrated
        with one-sample K-S tests using randomly generated events
        from a flat lightcurve (also compared to a flat lightcurve
        model) and comparing the distribution of the deviations thus
        found with the observed deviation.

history
        vinay kashyap (MMVI.X; modified from TIMEVARKS())
        bug: GTI filtering broken, tzero offsets incorrect, and
          FLAT_LC_CDF was changing TMIN,TMAX (VK; MMVII.IV)
```

B.259 TRANSLABEL [/mkemis]

```
function        translabel
        returns 2D array describing the levels involved in each transition
        array(0,*) are lower levels, and array(1,*) are upper levels

        if DR transitions, then a "(DR)" tag is appended to the lower level
        designations.

usage
        trans=translabel(chianti,wmn=wmn,wmx=wmx,/tex,config=config)

parameters
        chianti [INPUT; required] structure that contains all the relevant
                information gleaned from the CHIANTI database (type
                        DBHELP=RD_CHIANTI() & HELP,DBHELP,/STRUCTURE
                for a description of what this structure contains)

keywords
        wmn     [INPUT; default=0 Ang] only include wavelengths > WMN
        wmx     [INPUT; default=900 Ang] maximum wavelength to include
        tex     [INPUT] if set, returns the levels formatted for TeX
        config  [OUTPUT] returns the electron configurations of the lower
                and upper levels in 2D array
                * config(0,*) are for lower levels, and config(1,*) are
                  for upper levels
        _extra  [INPUT] junk -- ignore

history
        vinay kashyap (Nov96; based on LATEX_WVL_DEM.PRO of CHIANTI)
        made loops go over long integers (VK; Jun10)
```

B.260 TRIM [/external]

```
Project     : SOHO - CDS

Name        :
       TRIM()
Purpose     :
       Converts numbers to strings, without trailing zeros.
Explanation :
       Converts numbers into a string representation, and trims off leading
       and/or trailing blanks.  Differs from STRTRIM in that trailing zeros
       after the period are also trimmed off, unless NUMBER is already a
       string, or an explicit format is passed.
Use         :
       Result = TRIM( NUMBER  [, FORMAT ]  [, FLAG ] )
Inputs      :
       NUMBER  = Variable or constant.  May be of any ordinary including
                 string.  However, structures are not allowed.
Opt. Inputs :
       FORMAT  - Format specification for STRING function.  Must be a string
                 variable, start with the "(" character, end with the ")"
                 character, and be a valid FORTRAN format specification.  If
                 NUMBER is complex, then FORMAT will be applied separately to
                 the real and imaginary parts.

       FLAG    - Flag passed to STRTRIM to control the type of trimming:

                       FLAG = 0        Trim trailing blanks.
                       FLAG = 1        Trim leading blanks.
                       FLAG = 2        Trim both leading and trailing blanks.

                 The default value is 2.  If NUMBER is complex, then FORMAT
                 will be applied separately to the real and imaginary parts.

Outputs     :
       Function returns as a string variable representing the value NUMBER.
Opt. Outputs:
       None.
Keywords    :
       None.
Calls       :
       None.
Common      :
       None.
Restrictions:
```

```
       NUMBER must not be a structure.
       FORMAT must be a valid format specification, and must not be passed
               if NUMBER is of type string.
       FLAG must not be of string type, or an array.
Side effects:
       None.
Category    :
       Utilities, Strings.
Prev. Hist. :
       William Thompson        Applied Research Corporation
       May, 1987               8201 Corporate Drive
                               Landover, MD  20785

       William Thompson, Feb. 1992, added support for complex numbers, and
                                    fixed Unix problem with lowercase "e".
Written     :
       William Thompson, GSFC, May 1987.
Modified    :
       Version 1, William Thompson, GSFC, 9 April 1993.
               Incorporated into CDS library.
       Version 2, Zarro (SAC/GSFC), 3-Jun-98
               Added check for undefined input
      Version 3, Zarro (SM&A/GSFC), 1-Dec-99
              Returned invalid input as blank string
              to avoid downstream problems.
      Version 4, Zarro (SM&A/GSFC), 4-Jan-00
              Added /QUIET
      Version 5, Zarro (SM&A/GSFC), 20-Jan-00
              Vectorized
       Version 6, 24-Jan-2000, William Thompson, GSFC
               Fixed bug introduced in version 5.
       Version 7, 14-Mar-2000, Zarro (SM&A/GSFC)
               Moved check for unsupported type ahead of recursion
       Version 8, 22-Jul-2002, Zarro (LAC/GSFC)
               Made loop variables LONG
      Version 9, 08-Jan-2004, William Thompson, GSFC
              Added support for double-precision complex, and unsigned data
              types.
```

B.261 TWO_PHOTON_CH [/external]

```
PROJECT:  CHIANTI

      CHIANTI is an Atomic Database Package for Spectroscopic Diagnostics of
      Astrophysical Plasmas. It is a collaborative project involving the Naval
      Research Laboratory (USA), the University of Florence (Italy), the
      University of Cambridge and the Rutherford Appleton Laboratory (UK).


NAME:
       TWO_PHOTON_CH

PURPOSE:

       Calculate the 2 photon continuum from a hot, low density plasma.

      For the hydrogen isoelectronic sequence, A values
            Parpia, F. A., and Johnson, W. R., 1982, Phys. Rev. A, 26, 1142.
      and distribution function as a function of Z is taken from:
            Goldman, S.P. and Drake, G.W.F., 1981, Phys Rev A, 24, 183

      For the helium isoelectronic sequence, A values are from:
            Drake, G.W.F., 1986, Phys. Rev. A, 34, 2871
      and the distribution function as a function of Z is taken from:
            Drake, G.W.F., Victor, G.A., Dalgarno, A., 1969, Phys.
            Rev. A, 180, 25.
      in this paper the distribution is only given up to Z=10 but
      extrapolation to higher Z appears to be OK.

      Note that, unlike the freefree and freebound routines, two_photon
      requies the electron density to be specified. This is because there
      is a call to pop_solver

      To calculate the continuum spectrum the user should specify
      whether the plasma structure is specified through a differential
      emission measure (DEM) curve or through emission measure (EM)
      values. For the former, the DEM values are specified through the
      DEM_INT keyword and for the latter the EM values are specified
      through the EM_INT keyword. If neither are specified EM_INT
      values of unity are assumed.

      If DEM_INT values are specifed they must be specified on a
      temperature scale that has constant intervals in log10(T). EM_INT
```

```
      values do not have this restriction.

      Note that EM_INT must be used for isothermal plasmas.


CALLING SEQUENCE:

      TWO_PHOTON_CH,temperature, density, wavelength, intensity


INPUTS:

      TEMP    Temperature (in K). If the keyword /DT is set, then the
              temperatures must be specified with a fixed spacing in
              logT. E.g., TEMP=10.^[4.0,4.1,4.2].

      WAVELENGTH   Wavelengths in Angstroms. If /keV is set, then WVL is
                   assumed to contain energies in keV units.


OPTIONAL INPUTS:

      DENSITY       Electron density in cm^-3, can also be a 1D array
                    of the same size as Temperature. If there are several
                    temperatures specified but only one density, then
                    density is assumed the same at all temperatures.
                    If not specified, then default densities of 10^10
                    electrons/cm^3 are assumed.

      DEM_INT   This should be the same size as TEMP and contain
                differential emission measure values. Specifying
                DEM_INT places a restriction on TEMP (see above). Note
                that DEM_INT values are multiplied by the factor dT
                within the routine, whereas EM_INT values are not.

      EM_INT    This should be the same size as TEMP and contain
                emission measure values. Note
                that DEM_INT values are multiplied by the factor dT
                within the routine, whereas EM_INT values are not. If
                neither EM_INT or DEM_INT are specified, then a EM_INT
                vector of same size as TEMP with values of unity is
                assumed.

KEYWORD PARAMETERS:
```

```
         NO_SETUP:   If the procedure setup_elements has already been called
                     then the keyword /no_setup should be set to avoid
                     repeating this step

        MIN_ABUND:   If set, calculates the continuum only from those
                     elements which have an abundance greater than min_abund.
                     Can speed up the
                     calculations.  For example:
                     abundance (H)  = 1.
                     abundance (He) = 0.085
                     abundance (C)  = 3.3e-4
                     abundance (Si) = 3.3e-5
                     abundance (Fe) = 3.9e-5

        SUMT         If several temperatures have been specified, then /sumt
                     will sum the emissivities over the different
                     temperatures, giving an output INTENSITY that has the
                     same size as WAVELENGTH.

        PHOTONS      If set the continuum emissivity is output in photon
                     units rather than erg units.

        VERBOSE   If set, information is printed to screen as the routine runs.

        KEV       If set, then WVL is assumed to contain energies in keV units
                  rather than wavelengths in Angstroms.

OUTPUTS:

         RAD   2 photon continuum intensity in units of
              10^-40 erg cm^3/s/sr/Angstrom  per unit emission measure
              ( integral(N_H N_e dh) in cm^-5) if a DEM is not defined.

              If DEM values are defined, it is assumed that they are given
              as N_H N_e dh/dT.  The units are
              10^-40 erg/cm^2/s/sr/Angstrom

              If T is given as a 1-D array, then the output will be a 2-D
              array, with one element for each temperature and wavelength
              (but also see SUMT).

              If the keyword /keV is set, then the units of INT will be
              10^-40 erg cm^3/s/sr/keV
```

```
CALLS

      POP_SOLVER, SETUP_ION, SETUP_ELEMENTS, READ_MASTERLIST,
      CONVERTNAME

COMMON BLOCKS:

      ELEMENTS, ELVLC, WGFA, UPSILON, PROTON

EXAMPLE:

            > two_photon_ch,1.e+6,3.e+10,wvl,int
            > two_photon_ch,1.e+6,3.e+10,wvl,int,min_abund=3.e-5
            > two_photon_ch,1.e+6,3.e+10,wvl,int,/no_setup

PROGRAMMING NOTES

   For He 2-photon decays, the distribution function is from Table II
   of Drake et al. (1969), except that the values have been divided by
   the A-value from Drake (1986).


MODIFICATION HISTORY:
       Written by:     Ken Dere
       February 2001:  Version 1.0

      Ver.2, 19-Dec-2001, Peter Young
          Now allows an array of temperatures.
          Added /SUMT keyword.
          Added DEM_INT= optional input.
          Switched to using spl_init & spl_interp for the spline fits.
          Corrected a small bug.

      Ver.3, 20-Dec-2001, Peter Young
          Corrected a bug related to density indices.

      Ver.4, 23-Apr-2002, Peter Young
          Added /photons keyword.

      Ver.5, 28-May-2002, Peter Young
          Corrected way in which DEM_INT is handled.

      V. 6, 28-May-2002, Giulio Del Zanna (GDZ)
                  generalized directory concatenation to work for
                  Unix, Windows  and VMS.
```

```
            Corrected the description of the  units and various
            inaccuracies in the header.

      V.7, 14-Aug-02, GDZ
            Added keyword VERBOSE

      V.8,  4-Oct-2003, GDZ
                 modified the input to POP_SOLVER, now it is dealt with an
                 input structure.

      V.9,  8-Jun-2004, EL
                 modified the input to POP_SOLVER, now it includes ion/rec

      V.10, 5-Jul-2005
                 corrected problems with the input structure for pop_solver

      v.11 29-Jul-2005, GDZ
                 fixed bug, only define ioneq_ionrec when files are
                 there (it was failing with neutrals)

      v.12,  31-Aug-2005, GDZ
                 missed one correct definition of ioneq_ionrec. Fixed it now.

      v.13, 10-Mar-2006, Peter Young
                 added /keV keyword

      v.14, 12-Jun-2009, Enrico Landi
                 Changed the definition of the temperature array for ion fractions
                 in the IONREC variable, now taken directly from the output of
                 READ_IONEQ.PRO

      v.15, 11-Sep-2009, Peter Young
           Routine now checks that temperatures are evenly spaced
           when DEM_INT is specified. Also 0.1 dex spacing is no
           longer assumed.

      v.16, 6-Oct-2009, Peter Young
           Introduced EM_INT keyword for specifying emission measure
           values for isothermal plasmas.

VERSION     :   16,  6-Oct-2009

      V.PoA, 10-Aug-2010, Vinay Kashyap
             a la 25-Feb-2004, LiWei Lin
            Commented common block out and
```

```
Added _extra keyword
Added NeNh keyword to avoid calling proton dens
Exchanged popsol() for pop_solver for PoA convenience.
  popsol() puts a_value in required 2D format while
  for pop_solver this is done in a mother CHIANTI routine
  e.g. ch_synthetic.
Renamed routine two_photon_ch
```

B.262 ULIM_SIMPLEPOI [/stat]

```
function        ulim_simplepoi
        compute the upper limit to the intensity of an undetected source,
        given the background intensity, for a specified false detection
        probability alpha and detection probability beta, in the Poisson
        regime.

syntax
        ul=ulim_simplepoi(bkg,alpha,beta,alfa=alfa,Nthr=Nthr,verbose=v)

parameters
        bkg     [INPUT; required] background intensity over the source region
        alpha   [INPUT] maximum false detection probability
                * if not given, assumed to be 0.0027
                * if .GE.1, assumed to be given in units of equivalent
                  Gaussian sigma
                * if .LT.0, assumed to be given as 1./abs(ALPHA)
                * the actual false detection probability for the selected
                  counts threshold may be less than the specified value.
        beta    [INPUT] probability of detecting the source of given intensity
                * note that some authors use the notation beta to denote the
                  Type II error, the probability of a false negative, which is
                  1-beta in our notation.
                * if not given, assumed to 0.5
                * if .LT.0 or .GE.1, assumed to be given as 1.D-1.D/abs(BETA)

keywords
        alfa    [OUTPUT] ALPHA, as noted above, is the maximum allowed
                false detection probability.  Because of the discreteness
                of counts, the actual false det. probability that is used
                to figure out BETA will be different (though always less
                than ALPHA).  This computed probability is returned in this
                array.
        Nthr    [OUTPUT] the counts detection threshold chosen such that the
                false detection probability is < ALPHA
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

subroutines (all standard IDL)
        IGAMMA
        INTERPOL
        ERRORF

description
```

```
        Kashyap, van Dyk, Connors, Freeman, Siemiginowska, Xu, Zezas, 2010,
        "On Computing Upper Limits to Source Intensities", ApJ, 719, 900

        This encodes the simple Poisson case (footnotes 7 and 8) where the
        background intensity is known.  First, ALPHA, the false detection
        probability, is used to determine a counts threshold for detection,
        and that source intensity where the source would be detected with
        probability BETA is returned as the upper limit.

example usage
        bkg=2 & alpha=-1024.^2 & beta=0.5
        .run ulim_simplepoi

history
        Vinay Kashyap (Feb.MMX)
```

B.263 UNKINK [/util]

```
function        unkink
        given a curve with kinks in it, returns a curve by
        sweeping line segments across that smooths out kinks

syntax
        ys=unkink(y,n,xloc=xloc,/hires,nres=nres,/onestop,$
        verbose=verbose)

parameters
        y       [INPUT; required] the ordinate values of the curve to smooth
                * cannot handle multi-valued functions.  i.e., curve can be
                  concave upward or downward, but not leftward or rightward
        n       [INPUT; default=1] the number of points to cast on either
                side of a point to compute the smoothing line segment

keywords
        xloc    [INPUT] the abscissa values at which Y are defined
                * the default is to use lindgen(n_elements(Y))+1
                * this is useful only if the grid is non-uniform
                * if specified, but size does not match that of Y,
                  then it is ignored
        hires   [INPUT] if set, resamples the grid into a finer
                resolution (with NRES additional points in each
                segment), linearly interpolates Y onto this finer
                grid, and operates on the high-resolution curve
        nres    [INPUT] the number of additional points to include
                in each segment of Y(XLOC)
                * default is 1
        onestop [INPUT] if set, the smoothed curve is returned as
                the value interpolated on to XLOC by a line segment
                that is symmetric around XLOC
                * if not set, an average of the interpolated Ys is
                  computed from all the line segments that pass through
                  the given XLOC
        verbose [INPUT] controls chatter
        _extra  [JUNK] here only to prevent crashing the program

subroutines
        none

these are all the smoothing tools in PINTofALE
        ALALOESS() makes a loess curve
        CLTSMOOTH() removes -ves from background subtracted spectra or light curves
```

```
        CONV_RMF convolves with specified RMF
        HAARTRAN() smoothes by filtering on Haar wavelet coefficients
        LINEREM() removes lines from spectrum by iterative S/N filtering
        NOISMOOTH() does boxcar accumulation a la Ebeling's asmooth
        REGROUP() accumulates from one end
        SCRMF does fast convolution using optimized RMF
        SMOOTHIE does peak-descent accumulation to build up S/N
        SPLAC computes a segmented piecewise linear approximations
        UNKINK() removes sharp kinks from a curve
        VARSMOOTH() does local smoothing at specified scales
        VOORSMOOTH() does fast local smoothing at specified scales

example
        x=findgen(20) & y=abs(x-10) & plot,x,y,psym=-4
        oplot,x,unkink(y,10,/hires,nres=5),thick=2

history
        vinay kashyap (sep08)
```

B.264 UNLOGIT [/stat]

```
function        unlogit
        compute and return the inverse of the logit function.

        LOGIT() maps a variable in the [0,1] range to the
        real number line [-\infty,+\infty] by applying the
        transformation ln(x/(1-x)).
        conversely, UNLOGIT() maps the real number line to
        [0,1] by applying the transformation e^y/(1+e^y).

syntax
        x=unlogit(lx,sigl,sigx=sigx)

parameters
        lx      [INPUT; required] a variable in the range [-\infty,+\infty]
                * can be an array
                * value is set to NaN in output if input falls
                  outside valid range
        sigl    [INPUT] standard error on LX
                * if <0 and >-1, then ABS(SIGL) is taken to be the
                  fractional error
                * if <-1 and >-100, then ABS(SIGL) is taken to be
                  the percentage error
                * if <-100, then 1/ABS(SIGL) is taken to be the
                  fractional error

keywords
        sigx    [OUTPUT] standard error on UNLOGIT(X)
                * note that
                  SIGX = SIGL*exp(LX)/(1+exp(LX))^2 == SIGL*X/(1+exp(LX))
        _extra  [JUNK] here only to prevent crashing the program

history
        vinay kashyap (6jun05)
```

B.265 UPDATID [/]

```
function        updatid
        update the values in an ID structure and return the updated structure

syntax
        newid=updatid(idstr,fluxes,waves,emis,flxerr=flxerr,lambda=lambda,$
        spec=spec,bkg=bkg,/digit,verbose=v, width=width,sigma=sigma,$
        nsigma=nsigma,sigmay=sigmay,xsize=xsize,ysize=ysize,wid=wid,$
        dynrng=dynrng,markx=markx,marky=marky,markp=markp,markc=markc,$
        marks=marks,markl=markl,marko=marko,/legg,/quiet)

parameters
        idstr   [INPUT; required] ID structure to be updated
        fluxes  [I/O] if given, updates the IDSTR.(#).FLUX field
                * size must match number of components or number of IDs
                * if size does not match, but is given as a scalar or a
                  single-element vector, then
                  if +ve, treated as a global multiplicative factor
                  if -ve, the strongest line is normalized to abs(FLUXES)
                * if not single element, then gets overwritten on output
                  using IDSTR.(#).FLUX, so will be different if
                  DIGIT, LAMBDA, and SPEC are set
        waves   [INPUT] if given, updates the values of the observed
                wavelengths in IDSTR.WVL
                * size must match the number of components
                * if size does not match, but is given as a scalar or a
                  single-element vector, then taken to be wavelength shift
                * to update the wavelengths without messing with the fluxes,
                  call UPDATID() as
                  newid=updatid(idstr,1.,newwvl, ...)
        emis    [INPUT] if given, updates the emissivities in IDSTR.(#).EMIS
                * to update only the emissivities, call UPDATID() as
                  newid=updatid(idstr,1.,idstr.WVL,newemis, ...)

keywords
        flxerr  [I/O] if given, updates the errors in IDSTR by these
                values.  if not given, or is illegal, no changes are made.
                * overwritten on output using IDSTR.(#).FLUXERR
                * generally taken to be 1-sigma errors
                * size must match that of FLUXES
        lambda  [INPUT] wavelengths of base spectrum to be used to find
                the fluxes in case FLUXES is not given
        spec    [INPUT] spectrum to use to guess at the fluxes
                * size of SPEC and LAMBDA must match, else both are ignored
```

```
                * basically, at each IDSTR.WVL, find the total counts in a
                  small region nearby, add up the counts, and that's it.
                * SPEC and LAMBDA are used if
                  1: FLUXES is not given, OR
                  2: DIGIT is set
        bkg     [INPUT] if size matches that of SPEC, subtract from SPEC
                prior to finding the fluxes
                * if single element, assumed to be global background value
        digit   [INPUT] if set, then for each IDLSTR.WVL lets the user
                interactively pick the range in which to compute the fluxes
                * overrides FLUXES
        verbose [INPUT] controls chatter
        _extra  [INPUT] pass defined keywords to subroutines
                GETLOCMAX: WIDTH,SIGMA,NSIGMA
                PICKRANGE: SIGMA,XSIZE,YSIZE,WID,DYNRNG,MARKX,MARKY,MARKP,
                  MARKC,MARKS,MARKL,MARKO,LEGG,QUIET

subroutines
        GETLOCMAX
        PICKRANGE

history
        vinay kashyap (Dec98)
        added keyword FLXERR (VK; MarMM)
        added keyword VERBOSE; streamlined FLUXES/FLXERR output;
          converted to IDL5 notation (VK; JanMMI)
        added parameter EMIS (VK; JulMMVIII)
```

B.266 USE_CHIANTI [/external]

```
Project     : CHIANTI

      CHIANTI is an atomic database package for the calculation of
      continuum and emission line spectra from astrophysical plasmas. It is a
      collaborative project involving the Naval Research Laboratory
      (Washington DC, USA), the Arcetri Observatory (Firenze, Italy), and the
      Cambridge University (United Kingdom).


Name        : USE_CHIANTI

Purpose     : Sets up system variables for use of CHIANTI routines

Explanation : The  CHIANTI software uses system variables
              that have to be defined. This routine is called by the
              CHIANTI routines if these system variables are not defined.

              ** If one is using the solar-soft package, these should
              already be defined*****.

              ** If the CHIANTI package is used as stand-alone, then
              this routine can be used for the setup with e.g.:

              use_chianti,'/home/data/chianti/'

              if /home/data/chianti/ points to the actual place where
              the CHIANTI top directory is.


Use         : IDL> use_chianti, '/home/data/chianti/', $
                                  [ioneq= , abund=]

Inputs      : None

Opt. Inputs : The full pathname of the CHIANTI top directory.

Outputs     : None

Opt. Outputs: None

Keywords    : IONEQ  - to define the default ionization eq. file to be used.
              ABUND  -  to define the default abundance file to be used.
```

```
Calls       : None

Common      : None

Restrictions: None

Side effects: None

Category    :

Prev. Hist. : Based on use_dere_chianti, written by C D Pike, RAL

Written     : Giulio Del Zanna (GDZ)  DAMTP  (University of Cambridge, UK)

Modified    : Version 1, GDZ 10-Oct-2000
              Version 2, GDZ 10-Jan-2001
              added the definition of the !abund_file

              V. 3, GDZ, 2-May-2002 Modified substantially, adding a new
              system variable.

              V.4, 10-July-2002 GDZ
                Removed the definition of !chianti_top, upon request.

              V.5, 25-July-2002, GDZ
               Fixed a problem with IDL versions earlier than 5.3 (the routine
               would not compile). ALso, introduced the use of concat_dir for
               cross-platform compatibility.

              V.6, 06-Aug-02 GDZ
                Changed the use of CHIANTI system variables. Added comments on
                the which CHIANTI version is used.
              V.7 12-Aug-02 GDZ Changed default abundance file. Removed '***'
              V.8, GDZ 13-Feb-2003
                  Changed default ioneq  file, to include ALL the elements.

              V.9,  12-Aug-2005, GDZ
                  Changed default ioneq file from mazzotta_etal_ext to
                  mazzotta_etal_9.ioneq where ion fractions have been extended
                  up to 10^9 K. This is useful to avoid steps in the
                  emissivities at high-temperatures.

              V.10,  19-Sept-2005, GDZ
                  made last version backward-compatible.
```

```
              v.11, 3-Jun-2009, GDZ
                  made the new v.6 ioneq the default.

              v.12, 6-Jul-2012, Peter Young
                  in preparation for the v.7.1 release, I've
                  changed the default abundance file.

Version     : V.12,  6-Jul-2012
```

B.267 VARSMOOTH [/util]

```
function        varsmooth
        returns a locally smoothed array

        by default, smooths with a boxcar of specified scale.
        may also smooth with gaussians, lorentzians, beta-profiles,
        and asymmetric beta-profiles of varying widths; or obtain
        stepped averages over varying widths.

syntax
        sfunc=varsmooth(func,scale,xfunc=xfunc,weight=weight,type=type,$
        steps=steps,/gauss,/boxcar,nueff=nueff,nutilde=nutilde,$
        missing=missing,/fwhm,betap=betap,angle=angle)

parameters
        func    [INPUT; required] 1D array to be smoothed
                * if multi-D, gets converted to 1D
        scale   [INPUT] array containing the half-length scales over which
                FUNC must be smoothed.
                * if smaller in size than FUNC, fills out with last element
                * must be in bin units, unless
                  -- XFUNC and TYPE are both defined

keywords
        xfunc   [INPUT] x-coordinates for FUNC
                * size must match FUNC or must be bin boundaries on FUNC,
                else ignored.
                * ignored in case of boxcar smoothing
        weight  [INPUT] allows weighting of adjacent points while boxcar
                smoothing
                * if scalar, WEIGHT(*)=(dPIX)^(-WEIGHT(0))
        type    [INPUT] define the 3-parameter family function to be used
                to smooth
                * may be anything accepted by X3MODEL/MK_3MODEL, e.g.,
                  "Gaussian", "Lorentizan", "Beta=<value>",
                  "Slant=<angle>,<beta>", etc.
        steps   [INPUT] if set, ignores the actual values of SCALE
                and replaces FUNC in bins with contiguous stretches of
                SCALE with <FUNC> averaged over this contiguous stretch.
                * only those values of SCALE > STEPS[0] are considered
                  in the calculation -- those below it are held at one
                  bin each.
                * NUEFF and NUTILDE are not calculated for this case
        gauss   [INPUT] retained for backwards compatibility.  if set,
```

```
                and TYPE is not set, then sets TYPE='Gaussian'
        boxcar  [INPUT] if TYPE, GAUSS and STEPS are not set, the default
                is to carry out boxcar type smoothing.  however, the default
                there is to limit the smoothing scale to the maximum
                extent of the range, i.e., N(LSCAL)/2.  if BOXCAR is
                set, this limitation is ignored, and smoothing scales
                can be as large as possible.  however, note that this
                does not handle spillage over endpoints gracefully, so
                _THIS OPTION IS NOT RECOMMENDED_ unless you know what
                you are doing.
        nueff   [OUTPUT] the effective number of degrees of freedom
                in the problem, after smoothing.
                * the trace of the smoothing matrix
        nutilde [OUTPUT] correction to NUEFF
                * the trace of the product of the transpose of the smoothing
                  matrix and the matrix
                * dof ~ 2*NUEFF-NUTILDE
        _extra  [INPUT] pass defined keywords to
                MK_GAUSS: MISSING, FWHM
                MK_LORENTZ: BETAP, MISSING
                MK_SLANT: ANGLE, BETAP, MISSING

restrictions
        input array, FUNC, must be evenly sampled.
        requires subroutines:
                X3MODEL
                MK_3MODEL
                MK_GAUSS
                MK_LORENTZ
                MK_SLANT
                KILROY

these are all the smoothing tools in PINTofALE
        ALALOESS() makes a loess curve
        CLTSMOOTH() removes -ves from background subtracted spectra or light curves
        CONV_RMF convolves with specified RMF
        HAARTRAN() smoothes by filtering on Haar wavelet coefficients
        LINEREM() removes lines from spectrum by iterative S/N filtering
        NOISMOOTH() does boxcar accumulation a la Ebeling's asmooth
        REGROUP() accumulates from one end
        SCRMF does fast convolution using optimized RMF
        SMOOTHIE does peak-descent accumulation to build up S/N
        SPLAC computes a segmented piecewise linear approximations
        UNKINK() removes sharp kinks from a curve
        VARSMOOTH() does local smoothing at specified scales
```

```
        VOORSMOOTH() does fast local smoothing at specified scales

history
        vinay kashyap (Apr97)
        changed gaussian smoothing to wrap around (VK; May97)
        corrected bug of SCALE not filling out with last element (VK; Aug98)
        added keywords XFUNC, STEPS, and TYPE (latter makes GAUSS obsolete)
          (VK; MarMM)
        allowed XFUNC to be bin boundaries (VK/LL; Sep02)
        now variable rebinning takes effect only if scales are > STEPS[0]
          (VK; Feb03)
        added keywords NUEFF and NUTILDE (VK; Jan06)
        corrected NUTILDE calculation; added keyword BOXCAR (VK; Mar06)
```

B.268 VERNER_XS [/external]

```
PROJECT:  CHIANTI

     CHIANTI is an Atomic Database Package for Spectroscopic Diagnostics of
     Astrophysical Plasmas. It is a collaborative project involving the Naval
     Research Laboratory (USA), the University of Florence (Italy), the
     University of Cambridge and the Rutherford Appleton Laboratory (UK).

NAME

   VERNER_XS()

EXPLANATION

   Reads the Verner & Yakovlev (A&AS 109, 125, 1995) photoionization
   cross-section data and generates the values of the cross-section at
   the wavelengths WVL

INPUTS

   IZ     Atomic number of ion (e.g., 26 = Fe)

   ION    Spectroscopic number of ion (e.g., 13 = XIII)

   WVL    Wavelengths (in angstroms) for which cross-sections are
          required (1-D array).

OPTIONAL INPUTS

   DATA   By default VERNER_XS reads the Verner & Yakovlev data file when
          it is called. Through the DATA keyword the data array can be
          sent to VERNER_XS instead.

OUTPUT

   The photoionization cross-section for the ionization of the outer
   electron in units of mega-barns (Mb = 10^-18 cm^2) at the input
   wavelengths. E.g., for Fe XIII (ground configuration
   1s2.2s2.2p6.3s2.3p2) it is the cross-section for the removal of the
   3p electron.

HISTORY

   Ver.1, 24-Jul-2002, Peter Young
```

B.269 VOORSMOOTH [/util]

```
function        voorsmooth
        returns an array smoothed by convolving with appropriate function

        this is basically a front-end to IDL's CONVOL function, and is
        a faster alternative to VARSMOOTH (a VARSMOOTH that goes vrooom,
        if you will), smooths with gaussians, lorentzians, beta-profiles,
        asymmetric beta-profiles, etc.

syntax
        sy=voorsmooth(y,s,x,type=type,usekern=usekern,eps=eps,verbose=v,$
        /fwhm,betap=betap,angle=angle,missing=missing,/perbin)

parameters
        y       [INPUT; required] 1D array to be smoothed
        s       [INPUT; required] smoothing scale.  if an array, only
                the first element is used.
                * assumed to be in pixel units unless X is given
                * the meaning of the scale depends on the function
                  being used -- sigma for TYPE='Gauss' (unless FWHM
                  is set), half-width for TYPE='Lorentz', etc.
                * >>ignored<< if USEKERN is defined, but set to some
                  non-zero value anyway
        x       [INPUT; optional] values at which Y=Y(X) is defined
                * size must match Y
                  OR must be bin boundaries flanking Y
                  otherwise X is ignored
                * if given, S is assumed to be in the same units as X
                * if dX is non-uniform, Y and X are rebinned to the
                  finest possible grid and THEN convolved.

keywords
        type    [INPUT; default='Gaussian'] define the 3-parameter family
                function to be used to smooth
                * may be anything accepted by X3MODEL/MK_3MODEL, e.g.,
                  "Gaussian", "Lorentizan", "Beta=<value>",
                  "Slant=<angle>,<beta>", etc.
                * >>ignored<< if USEKERN is defined
        usekern [I/O] if defined legally on input, will ignore parameter
                S and keyword TYPE and use the defined kernel to smooth
                the input array Y.
        eps     [INPUT] a small number, 1e-6 by default
        verbose [INPUT] controls chatter
        _extra  [INPUT ONLY] use this to pass defined keywords to subroutines
```

```
                -- MK_GAUSS: FWHM, MISSING
                -- MK_LORENTZ: BETAP, MISSING
                -- MK_SLANT: ANGLE, BETAP, MISSING
                -- REBINW: PERBIN

restrictions
        calls subroutines:
                X3MODEL
                MK_3MODEL
                MK_GAUSS
                MK_LORENTZ
                MK_SLANT
                REBINW
                FINDEX

example
        make a "spectrum":
        y=fltarr(500) & y[250]=2.
        smooth with Gaussian of sigma=10 pix
        ys=voorsmooth(y,10.,type='Gauss',usekern=usekern)
        make a new kernel:
        kern=0.5*(shift(usekern,-30)+shift(usekern,30))
        smooth again:
        ys2=voorsmooth(y,10.,type='Gauss',usekern=kern)
        display:
        plot,y & peasecolr & oplot,ys,color=2 & oplot,ys2,color=3

these are all the smoothing tools in PINTofALE
        ALALOESS() makes a loess curve
        CLTSMOOTH() removes -ves from background subtracted spectra or light curves
        CONV_RMF convolves with specified RMF
        HAARTRAN() smoothes by filtering on Haar wavelet coefficients
        LINEREM() removes lines from spectrum by iterative S/N filtering
        NOISMOOTH() does boxcar accumulation a la Ebeling's asmooth
        REGROUP() accumulates from one end
        SCRMF does fast convolution using optimized RMF
        SMOOTHIE does peak-descent accumulation to build up S/N
        SPLAC computes a segmented piecewise linear approximations
        UNKINK() removes sharp kinks from a curve
        VARSMOOTH() does local smoothing at specified scales
        VOORSMOOTH() does fast local smoothing at specified scales

history
        vinay kashyap (2001Jul; derived from varsmooth, but really
          a precursor, hence the [Noordwijk influenced] name)
```

```
bug correction oo v/s o0; now reverse kernel before convol (VK; Oct01)
allowed X to be bin boundaries (VK/LL; Sep02)
accounted for 1-pix shift due to centering of kernel (VK; May03)
```

B.270 VRDB [/util]

```
procedure       vrdb
        print selected rows and columns from RDB table (see RDB.PRO)

syntax
        vrdb,dbstr,pick,cols=cols,/row,wfield=wfield

parameters
        dbstr   [INPUT; required] structure as read in by RDB.PRO
        pick    [INPUT] array of position indices to print out
                * default is [0]

keywords
        cols    [INPUT] array of column indices to print out
                * default is to print all except DBSTR.HEAD
                * if string array, then assumed to contain column names
                  -- names do not have to be exact matches
        row     [INPUT] if set, prints out the fields in a row
                * default is to print them out in a column.
                * -ve values will force all possible rows to be output
        wfield  [INPUT] array of integers specifying the width at which
                each column must be printed
                * if not specified, tacks on +2 to the lengths deduced
                  from the very first row to be printed
        _exra   [JUNK] here only to prevent crashing the program

history
        vinay kashyap (FebMM)
```

B.271 WC [/util]

```
function        wc
        obtains the number of lines, words or characters in the
        specified file, using the UNIX wc command

syntax
        lines = wc(files,/word,/char,comm=comment,mat=mat,/h)

parameters
        files   [INPUT; required] name of file

keywords
        word    [INPUT] if set, returns number of words
        char    [INPUT] if set, returns number of characters
        comm    [INPUT] excludes lines where the first character is
                        the comment character, as specified
        mat     [INPUT] operate only on text containing matching string
        _extra  [JUNK] here only to prevent crashing the program

notes
        if input is an array, the output is also an array, but
        with one extra element at the end, containing the total
        number of lines, words, or characters found.

restrictions
        IDL wrapper to the UNIX wc command, so obviously works only for UNIX.

history
        vinay kashyap (3/25/93)
```

B.272 WEE_PARAMESRAN [/util]

```
function        wee_paramesran,img
        computes and returns the sharpness of an image
        (see Wee, C.-Y., and Paramesran, R., 2008, ICSP 2008)
        http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&tp=&arnumber=4697259

parameters
        img     [INPUT; required] 2D image

keywords
        firstk  [INPUT] how many of the eigenvalues to include in
                measure of sharpness
                * default is to include all of them
        verbose [INPUT] controls chatter
        _extra  [JUNK] here to prevent program from crashing

history
        vinay kashyap (2014jul)
```

B.273 WHEE [/util]

```
procedure       whee
        emulates a wheel rolling in place by printing the characters
        "|", "/", "-", "\", "|", "/", "-", "\" in succession

syntax
        whee,spoke=spoke,/moveit

parameters      NONE

keywords
        spoke   [I/O] a +ve integer that on input says at what state the
                spoke in the wheel is in.  if not given, set to 1.
                contains this value on output (see MOVEIT)
                * if > 8, wraps back to 1
        moveit  [INPUT] if set, increments SPOKE by 1 prior to output

usage
        IDL> for i=0,3000 do whee,spoke=spoke,/moveit

history
        vinay kashyap (Feb97)
```

B.274 WHEREIS [/util]

```
function        whereis
        apes the csh command ‘which‘ in the IDL environment, and
        finds the path to the specified file by looking through
        all the directories specified in the !PATH variable.

        NOTE: it is possible to use ROUTINE_INFO to get the path
        to a compiled subroutine directly:
          help,routine_info(routine,/source,/function),/structure
        but clearly you need to know the exact name of the routine
        (can’t use wildcards) as well as whether what you need is
        a procedure or a function beforehand.  also, they have to
        be compiled procedures, not, for example, command files or
        scripts or datafiles lurking under the horizon.  and what
        if there are duplicates in different directories?

        btw, FILEPATH() is completely uselss for this purpose and
        FINDFILE() is of limited use because you still need to
        explicitly say which directory to look in.

syntax
        fpath=whereis(file,lookin=lookin,/dironly,/nospawn,verbose=verbose)

parameters
        file    [INPUT; required] name(s) of file to search for
                * may include shell wildcards, regexps, etc.
                  (but remember to prepend the escape character "\")
                * may also be an array.
        path    [OUTPUT] full path to the requested file(s)
                * there is no reason to expect that the array sizes of
                  FILE and PATH will match.

keywords
        lookin  [INPUT] path specification in which to look for FILE
                * this will be passed without comment to EXPAND_PATH()
        dironly [INPUT] if set, returns only the name(s) of the
                directory(ies) containing the specified file(s)
        nospawn [INPUT] if set, acts as wrapper for FINDFILE()
                even on UNIX systems, i.e., does spawn find
        verbose [INPUT] if set, spits out informational messages
        _extra  [JUNK] here only to prevent crashing the program

notes
        spawns "find" on UNIX systems, and calls FINDFILE() otherwise
```

```
history
        vinay kashyap (Dec2001)
```

B.275 WHICH [/obsolete]

```
procedure       which
        apes the csh command ‘which‘ in the IDL environment, and
        finds the path to the specified file by looking through
        all the directories specified in the !PATH variable.

        NOTE: it is possible to use ROUTINE_INFO to get the path to a
        compiled subroutine directly:
          help,routine_info(routine,/source,/function),/structure
        but clearly you need to know the exact name of the routine
        (can’t use wildcards) as well as whether what you need is
        a procedure or a function beforehand.  also, they have to
        be compiled procedures, not, for example, command files or
        scripts or datafiles lurking under the horizon.  and what
        if there are duplicates in different directories?
        btw, FILEPATH is completely uselss for this purpose.

syntax
        which,file,path,/findall,/dironly,verbose=verbose,$
        /all,outfile=outfile,search=search,quiet=quiet

parameters
        file    [INPUT; required] name(s) of file to search for
                * may include shell wildcards, regexps, etc.
                  (but remember to prepend the escape character "\")
                * may also be an array.
        path    [OUTPUT] full path to the requested file(s)
                * there is no reason to expect that the array sizes of
                  FILE and PATH will match.

keywords
        findall [INPUT] if set, continues to search even if
                the file has been found
                * if not set, and FILE is scalar, stops searching
                  as soon as at least one copy of file has been found
                * set automatically if FILE is an array or contains
                  special characters ’*’, ’?’, ’[’, ’]’, ’\’
                - in this case, set explicitly to 0 to quit at first
                  match.
        dironly [INPUT] if set, returns only the name(s) of the
                directory(ies) containing the specified file(s)
        verbose [INPUT] if set, spits out informational messages
        all     [INPUT] added for compatibility with SSW version,
                same as FINDALL
```

```
        outfile [OUTPUT] added for compatibility with SSW version,
                same as PATH
        search  [INPUT] added for compatibility with SSW version,
                automatically sets FINDALL and tacks on '*'s to FILE
        quiet   [INPUT] added for compatibility with SSW version,
                same as setting VERBOSE=0
        _extra  [JUNK] here only to prevent crashing the program

restrictions
        FOR UNIX ONLY (spawns find)

history
        vinay kashyap (FebMM)
        quits if not UNIX (VK; FebMMI)
        added keywords ALL, OUTFILE, and SEARCH for SSW-IDL compatibility
          (VK; May03)
```

B.276 WRT_LN_APED [/mkemis]

```
procedure       wrt_ln_aped
        writes out line intensities computed from an APED-type
        database in a form that is easier to access.

usage
        wrt_ln_aped,apedfil,logT,outdir=outdir,apeddir=apeddir

parameters
        apedfil [INPUT; required] FITS file, output of APEC
                * passed w/o comment to RD_APED
        logT    [INPUT] log_10(T[K]) at which to store the emissivities
                * default is FINDGEN(81)*0.05+4

keywords
        outdir  [INPUT; default: aped] directory tree under which to
                store the output files
                * WARNING: do not append a "/" (will crash if N_E is set)
        apeddir [INPUT] directory containing APEDFIL
                * passed w/o comment to RD_APED
        abund   [INPUT] APED includes abundances, so "undo" by dividing
                by the abundances
                * default is to use Anders & Grevesse

        _extra  [JUNK] here only to avoid crashing the program

side effects
        deposits (possibly large) files in OUTDIR
        * ATOM_wvl      wavelengths of all the transitions
        * ATOM_tem      temperatures
        * ATOM_##.#     line intensities I(tem,wvl)@logP
        * ATOM_ion      ionic states corresponding to wvl
        * ATOM_src      source of line information
                        3: APED
        * ATOM_lvl      level designations of each transition
        * ATOM_ecn      e-configurations of each level

restrictions
        requires APED-style datafile
        subroutines: RD_APED, KILROY, INICON

history
        vinay kashyap (AprMM)
```

B.277 WRT_LN_CHIANTI [/mkemis]

```
procedure       wrt_ln_chianti
                writes out line intensities computed from a CHIANTI-type
                database in a form that is easier to access.

usage
        wrt_ln_chianti,logP,logT,Z=Z,N_e=N_e,outdir=outdir,$
        wmn=wmn,wmx=wmx,istate=istate,chidir=chidir

parameters
        logP    [INPUT; required] log10(Pressure [cm^-3 K])
        logT    [INPUT] array of log10(Temperature [K]) at which to compute
                line intensities.  default: logT=findgen(81)*0.05+4.
                * best keep delta(logT) constant

keywords
        Z       [INPUT; default: ALL] element, ionic state (e.g.:'He','Fe XII')
        n_e     [INPUT] if set, computes line emissivities at the
                specified electron density [cm^-3]
                * overrides LOGP
                * appends a "D" to OUTDIR
        outdir  [INPUT; default: emisschianti] directory tree under which to
                store the output files
                * WARNING: do not append a "/" (will crash if N_E is set)

        _extra  [INPUT ONLY] allows passing defined keywords (WMN,WMX,N_E,
                ISTATE,CHIDIR) to LINE_CHIANTI

side effects
        deposits (possibly large) files in OUTDIR
        * ATOM_wvl      wavelengths of all the transitions
        * ATOM_tem      temperatures
        * ATOM_##.#     line intensities I(tem,wvl)@logP
        * ATOM_ion      ionic states corresponding to wvl
        * ATOM_jon      ionic states that matter to the ion balance
        * ATOM_src      source of line information
                        2: CHIANTI
        * ATOM_lvl      level designations of each transition
        * ATOM_ecn      e-configurations of each level

restrictions
        requires CHIANTI-style database
        requires LINE_CHIANTI and associated subroutines
        input LOGP must be uniquely writeable in format f4.1 (e.g.,14.5)
```

```
history
        vinay kashyap (Nov96)
        forced output filenames to not have spaces, changed output
          file names and format, esp. of e-configurations (VK; Dec96)
        changed ATOM_src CHIANTI code from 0 to 2 (VK; Jan96)
        added keyword N_E (VK; Jun97)
        modified to account for dielectronic recombination lines in
         CHIANTIv3 (VK; OctMM)
```

B.278 WRT_LN_GENERIC [/mkemis]

```
procedure       wrt_ln_generic
        writes out line intensities to a PoA style database from
        a PoA style line emissivities structure.

usage
        wrt_ln_generic,linstr,outdir,tag,abund=abund,excieq=excieq,$
        eps=eps,chidir=chidir,eqfile=eqfile

parameters
        linstr  [INPUT; required] line emissivities structure, of
                the same form output by RD_LINE()
                * NOTE: the emissivities must be in units of
                  [1e-23 ergs cm^3/s]
        outdir  [INPUT; required] directory to deposit the database files
                * WARNING: the PINTofALE convention is to automatically
                  append a "D" while reading from emissivities derived
                  at constant density.  so if the emissivities being
                  written out were derived at constant density, then
                  please makes sure that OUTDIR has an extra "D" at the
                  end.  this is not checked for, because a priori this
                  program can have no idea of the nature of the emissivities
                  being input.
        tag     [INPUT; required] a value (such as pressure, or density)
                defining the emissivities.  see description of output
                files below for where it is used.
                * MUST BE writeable in the format (f4.1)
                * if > 99, the alog10 value is used
                * if < 0, forced to be 00.0

keywords
        abund   [INPUT] if given, assumes that the input emissivities
                have been multiplied by these abundance values, and
                that you don't want this factor stored in the database
                files, and divides them out.
        excieq  [INPUT] if set, assumes that the input emissivities have
                been multiplied by the ion balance defined by the
                keywords CHIDIR and EQFILE, and that you don't want them
                to be included in the stored database, and divides them
                out over the range where the ion balance is > EPS
        eps     [INPUT] a small number
                * default is 1e-6
        verbose [INPUT] controls chatter
        _extra  [INPUT ONLY] pass defined keywords CHIDIR and EQFILE
```

```
                to RD_IONEQ()

side effects
        deposits (possibly large) files in OUTDIR
        * ATOM_wvl      wavelengths of all the transitions
        * ATOM_tem      temperatures
        * ATOM_##.#     line emissivities [tem,wvl]_TAG
        * ATOM_ion      ionic states corresponding to wvl
        * ATOM_jon      ionic states that matter to the ion balance
        * ATOM_src      source of line information
        * ATOM_lvl      level designations of each transition
        * ATOM_ecn      e-configurations of each level

subroutines:
        ARRAYEQ
        INICON
        SETSYSVAL, LEGALVAR()
        RD_IONEQ(), READ_IONEQ

history
        vinay kashyap (Jun02)
        Antonio Maggio discovered that ion, jon, and src files were
          writing out float rather than long (VK; Aug02)
```

B.279 WRT_OGIP_RMF [/util]

```
procedure       wrt_ogip_rmf
        write out a response matrix into an OGIP-compatible FITS file

syntax
        wrt_ogip_rmf,rmf,rmfile,hdrfil=hdrfil,matext=matext,ebext=ebext,$
        telescop=telescop,instrume=instrume,filter=filter,detnam=detnam,$
        grating=grating,chantype=chantype,detchans=detchans,$
        hduclass=hduclass,hduclas1=hduclas1,hduclas2_rm=hduclas2_rm,$
        hduclas2_eb=hduclas2_eb,hduclas3=hduclas3,hduvers=hduvers,$
        tlmin4=tlmin4,numgrp=numgrp,numelt=numelt,lo_thres=lo_thres,$
        /ccls,/ccnm,/cdtp,cvsd=cvsd,cvst=cvst,cdes=cdes,$
        rmfversn=rmfversn,hduvers1=hduvers1,hduvers2=hduvers2,$
        origin=origin,creator=creator,revision=revision,mission=mission

parameters
        rmf     [INPUT; required] response matrix structure, in the
                same format as is read in by, e.g., RD_OGIP_RMF()
        rmfile  [INPUT; required] name of output file

keywords
        hdrfil  [INPUT] name of existing OGIP-style response matrix file
                from which to copy the headers as far as possible

                the following are header keywords recommended by OGIP,
                along with a brief description of what they mean.  the
                default values used by this program are also listed.
                see http://legacy.gsfc.nasa.gov/docs/heasarc/caldb/docs/memos/cal_gen_92

        matext          = 'MATRIX'      /MATRIX or SPECRESP MATRIX, name of extension
        ebext           = 'EBOUNDS'     /name of extension
        telescop        = 'TELESCOP'    /telescope used
        instrume        = 'INSTRUME'    /instrument used
        filter          = 'NONE'        /the instrument filter in use (if any)
        detnam          = 'DETNAM'      /detector used
        grating         = 'GRATING'     /grating used (if any)
        chantype        = 'PHA'         /PHA or PI, uncorrected or corrected for gain
        detchans        = n_elements(rmf.EMN)   /number of raw detector channels
        hduclass        = 'OGIP'        /file format is OGIP standard
        hduclas1        = 'RESPONSE'    /extension contains response data
        hduclas2_rm     = 'RSP_MATRIX'  /extension contains a response matrix
        hduclas2_eb     = 'EBOUNDS'     /extension contains a response matrix
        hduvers         = '1.3.0'       /version of the file format
        tlmin4          = rmf.FIRSTCHAN /first channel in the response
```

```
        numgrp          = total(rmf.N_GRP)      /number of channel subsets
        numelt          = total(rmf.N_CHAN)     /number of response elements
        lo_thres        = 0.000001      /lower threshold used to construct the matrix
        hduclas3        = 'DETECTOR'    /REDIST (RMF) or DETECTOR (with QE, no ARF) or FULL (every
        rmfversn        = '1992a'       /obsolete
        hduvers1        = '1.1.0'       /obsolete
        hduvers2        = '1.2.0'       /obsolete
        ccls            = 'CPF'         /the OGIP-class of this calibration file
        ccnm            = 'MATRIX' or 'EBOUNDS' /CIF codename for this type of calibration dataset
        cdtp            = 'DATA'        /OGIP code for the form of the contents of the file
        cvsd            = YYYY-MM-DD    /UTC date of beginning applicability
        cvst            = HH:MM:SS      /UTC time of beginning applicability
        cdes            = 'RSP'         /brief descriptive summary of this dataset
        origin          = whoami@host   /source of FITS file
        creator         = 'PINTofALE'   /program creating this file
        revision        = 1
        mission         = 'MISSION'     /mission

        _extra  [JUNK] here only to prevent crashing the program

restrictions
        * requires the IDL-Astro library
        * if not a UNIX system, keyword ORIGIN must be explicitly set

side-effects
        writes out a FITS file to disk

history
        vinay kashyap (Nov2001)
        force variable length arrays even when N_GRP=1 (VK; Jul02)
```

B.280 WVLT_SCALE [/util]

```
function        findscale
        returns the lengthscale in pixels at each point on the given curve
        using the MexHat Wavelet Transforms as a guide

syntax
        ls=wvlt_scale(curve,/coarse)

parameters
        curve   [INPUT; required] regularly gridded 1D array of function
                values to be used to compute the length scales
                * if scalar, returns 0
                * if >1D, convert to 1D

keywords
        coarse  [INPUT; default=2] explore scales at specified coarseness
                e.g.: if COARSE=1, every scale from 1 to N(CURVE)-1 will
                be tested
        _extra  [INPUT] junk; ignore.  here only to prevent crashing program.

history
        vinay kashyap (Apr97)
        fixed bug when N(CURVE) was a power of 2 (LiWei Lin/VK; Jan'03)
```

B.281 X3MODEL [/fitting]

```
procedure       x3model
        procedure written to be compatible with FIT_LEVMAR, and also
        with IDL's CURVEFIT, GHRS-IDL's WFIT, etc.
        computes F(X;A) and partial derivatives as needed for a
        combination of 3-parameter family of curves

syntax
        x3model,x,a,f,pder, group=group,delp=delp,type=type,/asis,/allcomp,$
        missing=missing,/fwhm,/norm,betap=betap,vrot=vrot

parameters
        x       [INPUT; required] points at which to generate models
        a       [INPUT; required] array of parameter values
                * N_ELEMENTS(A) must be a multiple of 3
                * expected to be sequential triples of (position, width, height)
        f       [OUTPUT; required] output f=f(x;a)
        pder    [OUTPUT; optional] partial derivatives for each parameter

keywords
        rmfstr  [INPUT] structure w/ OGIP standard rmf with which to
               convolve model. SEE: RD_OGIP_RMF.PRO
        x_seg   [INPUT] x-axis wavelength grid on which to construct
                the model spectrum; used when RMFSTR is set, and X is
                then ignored
       _extra   [INPUT] pass defined keywords to subroutines
                MK_3MODEL: GROUP, DELP, TYPE, ASIS, ALLCOMP
                MK_GAUSS: MISSING, FWHM, NORM
                MK_LORENTZ: BETAP, MISSING, NORM
                MK_SLANT: ANGLE, BETAP, MISSING, NORM
                MK_ROGAUSS: VROT, FWHM, NORM, MISSING
                MK_SINUSOID: PHASE, MISSING

requires
        MK_3MODEL
        MK_GAUSS
        MK_LORENTZ
        MK_SLANT
        MK_ROGAUSS
        MK_POWLAM
        MK_SINUSOID
        SCRMF
        REGROUP_RMF
```

```
history
        vinay kashyap (Oct98; modified from GMODEL.PRO)
        added MK_SLANT (VK; MarMM)
        added MK_ROGAUSS,MK_POWLAM (VK; JunMM)
        added keywords RMFSTR and X_SEG and call to SCRMF (LL; Aug03)
        added MK_SINUSOID (VK; Sep04)
```

B.282 X3MODEL_F [/fitting]

```
function        x3model_f
        function written to be compatible with MPFIT, and is
        identical in all respects to X3MODEL, except that this
        is a function and not a procedure.
        returns F(X;A) and computes partial derivatives as needed
        for a combination of 3-parameter family of curves

syntax
        f=x3model_f(x,a,pder, group=group,delp=delp,type=type,/asis,/allcomp,$
        missing=missing,/fwhm,/norm,betap=betap,vrot=vrot)

parameters
        x       [INPUT; required] points at which to generate models
        a       [INPUT; required] array of parameter values
                * N_ELEMENTS(A) must be a multiple of 3
                * expected to be sequential triples of (position, width, height)
        pder    [OUTPUT; optional] partial derivatives for each parameter

keywords
        _extra  [INPUT] pass defined keywords to subroutines
                MK_3MODEL: GROUP, DELP, TYPE, ASIS, ALLCOMP
                MK_GAUSS: MISSING, FWHM, NORM
                MK_LORENTZ: BETAP, MISSING, NORM
                MK_SLANT: ANGLE, BETAP, MISSING, NORM
                MK_ROGAUSS: VROT, FWHM, NORM, MISSING
                MK_SINUSOID: PHASE, MISSING

requires
        MK_3MODEL
        MK_GAUSS
        MK_LORENTZ
        MK_SLANT
        MK_ROGAUSS
        MK_POWLAM
        MK_SINUSOID

history
        converted from X3MODEL (VK/LL; Oct02)
        added MK_SINUSOID (VK; Sep04)
```

B.283 XSPECPLOT [/util]

```
procedure       xspecplot
        plot two curves in two panels the way XSPEC does spectra and
        residuals

syntax
        xspecplot,x,y1,y2,y1err=y1err,y2err=y2err,xerr=xerr,$
        y1over=y1over,y2over=y2over,$
        fraclow=fraclow,verbose=verbose,outfile=outfile,$
        Ulstyl=Ulstyl,Upsym=Upsym,Ucol=Ucol,Uylog=Uylog,Uytitle=Uytitle,Uyr=Uyr,$
        Llstyl=Llstyl,Lpsym=Lpsym,Lcol=Lcol,Lylog=Lylog,Lytitle=Lytitle,Lyr=Lyr,$
        oUlstyl=oUlstyl,oUpsym=oUpsym,oUcol=oUcol,oLlstyl=oLlstyl,oLpsym=oLpsym,oLcol=oL
        [PLOT keywords]


parameters
        x       [INPUT; required] abscissa array
        y1      [INPUT; required] array that goes in the top panel
        y2      [INPUT; required] array that goes in the bottom panel
                * sizes of all arrays _must_ match

keywords
        y1err   [INPUT] error bar on Y1
        y2err   [INPUT] error bar on Y2
        xerr    [INPUT] error bar on Y2
                * if scalar and +ve, assumed to be a fractional error
                  if between 0 and 1, and %error if greater than 1
                * if scalar and -ve, abs value is assumed to be a constant
                  error
                * if array of same size as X, assumed to be symmetric
                * if array of size [2,NX] or [NX,2], assumed to be upper
                  and lower bounds
                * ignored if array size doesn't match X
        y1over  [INPUT] additional arrays to overplot in upper panel
        y2over  [INPUT] additional arrays to overplot in lower panel
                * size must be [m,NX] or [NX,m]
                * if not set, both assumed to be 0*NX
        fraclow [INPUT] what fraction of the plot should the lower panel
                take up?
                * default is 0.35
                * hard minimum of 0.05 and hard maximum of 0.95
        verbose [INPUT] controls chatter
        outfile [INPUT] if set, makes a postscript plot
                * if set to a string, assumes that that is the name of
```

```
                the file, otherwise writes to ./xspecplot.ps
        Ulstyl  [INPUT] line style for upper panel
        Llstyl  [INPUT] line style for lower panel
        linestyle       [INPUT] caught and discarded
        Upsym   [INPUT] PSYM for upper panel
        Lpsym   [INPUT] PSYM for lower panel
        psym    [INPUT] caught and discarded
        Ucol    [INPUT] COLOR for upper panel
        Lcol    [INPUT] COLOR for lower panel
        color   [INPUT] caught and discarded
        Uylog   [INPUT] YLOG for upper panel
        Lylog   [INPUT] YLOG for lower panel
        ylog    [INPUT] caught and discarded
        Uytitle [INPUT] YTITLE for upper panel
        Lytitle [INPUT] YTITLE for lower panel
        ytitle  [INPUT] caught and discarded
        Uyr     [INPUT] YRANGE for upper panel
        Lyr     [INPUT] YRANGE for lower panel
        oUlstyl [INPUT] line styles of Y1OVER (must be array of size M)
        oLlstyl [INPUT] line styles of Y2OVER (must be array of size M)
        oUpsym  [INPUT] PSYM for Y1OVER (must be array of size M)
        oLpsym  [INPUT] PSYM for Y2OVER (must be array of size M)
        oUcol   [INPUT] COLOR for Y1OVER (must be array of size M)
        oLcol   [INPUT] COLOR for Y2OVER (must be array of size M)
        yrange  [INPUT] caught and discarded

        _extra  [INPUT ONLY] pass defined keywords to PLOT

history
        vinay kashyap (2012-jun)
```

B.284 XSPLOT [/util]

```
procedure       xsplot
        make XSPEC like double-panel plots for given counts spectrum
        and associated model

syntax
        xsplot,data,model,egrid,dataerror,linecolor,/perbin,minct=minct,$
        dataN=dataN,modN=modN,eeN=eeN,dsigN=dsigN,lcolN=lcolN,$ ;N=0..9
        /resid,/fracres,/delchi,/ratio,verbose=vv,wsplit=wsplit,$
        /ylog,xtitle=xtitle,ytitle=ytitle,xthick=xthick,ythick=ythick,$
        title=title,subtitle=subtitle,charsize=charsize,charthick=charthick,$
        yrange=yrange,zrange=zrange, /xlog,xrange=xrange,thick=thick,$
        /halfit,eps=eps, /nuthin,/noday,/notime,/nouser,/nopack,stacol=stacol

parameters
        data    [INPUT; required] array of observed counts
        model   [INPUT] array of model counts
                * must match size of DATA
                - extra bins are ignored
                - missing bins are assumed to be 0
        ee      [INPUT] x-axis grid defining DATA and MODEL
                * if size is N(DATA), assumed to be mid-bin values
                * if size is N(DATA)+1, assumed to be bin boundaries
                * if not given, 1-based integer indices are used
        dsig    [INPUT] error bars on DATA
                * if not given, assumed to be Gehrels' approximation
                  of Poisson counts error, sqrt(DATA+0.75)+1
                * if set to a scalar, then
                  - if -ve, errors are assumed constant = abs(DSIG)
                  - if integer, assumed to be in units of SIGMAs, i.e.,
                    DSIG*default are plotted
                  - if not integer and 0<DSIG<0.5, then assumed to be a
                    fractional error DSIG*abs(DATA)
                  - if not integer and 0.5<DSIG<10, then assumed to be
                    in units of SIGMAs, i.e., DSIG*default are plotted
                  - if not integer and DSIG>10, assumed to be a percentage
                    error, (DSIG/100)*abs(DATA)
        lcol    [INPUT] 3-element byte array specifying the line colors
                  for MODEL, DATA, DSIG in that order
                * if incompletely given, missing elements are assumed
                  to be [0,10,20] or [255,245,235] depending on whether
                  !p.background is 255 or 0.

keywords
```

```
perbin  [INPUT] if set, will treat the input as counts/bin
        and plot accordingly.  if not set, will compute the
        bin widths and plot the density instead
        * if EE is not given, the bin widths will be 1 and
          this has no relevance except to the YTITLE
minct   [INPUT] if given, regroups the input DATA on the fly
        to have a minimum number of counts in each bin, and
        then regroups all the other arrays to the same grid
        * forces PERBIN to be ignored
verbose [INPUT] controls chatter
dataN   [INPUT] for N=0..9, additional DATA arrays
        * size must match DATA
modN    [INPUT] for N=0..9, additional MODEL arrays
        * size must match DATA
eeN     [INPUT] for N=0..9, additional EE arrays
        * if not given, default is to use EE
dsigN   [INPUT] for N=0..9, additional DSIG arrays
        * size must match DATA
lcolN   [INPUT] for N=0..9, line colors, by default, N*25+[25,35,45]
resid   [INPUT] if set, plots (DATA-MODEL) in lower panel
fracres [INPUT] if set, plots (DATA-MODEL)/MODEL in lower panel
delchi  [INPUT] if set, plots (DATA-MODEL)/DSIG in lower panel
ratio   [INPUT] if set, plots DATA/MODEL in lower panel
        * if more than one of RESID,FRACRES,DELCHI,RATIO are set,
          the priority is in that order.  i.e., the later ones
          in the list override anything given before it.
        * if none are set, assumes /DELCHI
wsplit  [INPUT] fraction by which to split the upper and lower windows
        * default is 0.35
        * ignored if <0.05 or > 0.95
ylog    [INPUT] catch and release to upper PLOT
xtitle  [INPUT] catch and release to lower PLOT
ytitle  [INPUT] catch and release to upper PLOT
xthick  [INPUT] catch and release to PLOT
ythick  [INPUT] catch and release to PLOT
title   [INPUT] catch and release to upper PLOT
charsize [INPUT] catch and release to PLOT
charthick [INPUT] catch and release to PLOT
subtitle [INPUT] catch and release to lower OPLOT
yrange  [INPUT] catch and release to upper PLOT
zrange  [INPUT] yrange catch and release to lower PLOT
xstyle  [INPUT] catch and discard
psym    [INPUT] catch and discard
color   [INPUT] catch and discard
_extra  [INPUT ONLY] pass defined keywords to subroutines:
```

```
                PLOT: XLOG, XRANGE, THICK
                STAMPLE: NUTHIN, NODAY, NOTIME, NOUSER, NOPACK, STACOL
                MID2BOUND: HALFIT, EPS

examples
        xsplot,data,model
        xsplot,data,model,/verbose      ;to stamp creator name and time
        xsplot,data,model,wvl           ;to include physical x-axis grid
        xsplot,data,model,wvl,0         ;to not plot error bars
        xsplot,counts,model,minct=25    ;adaptively group the counts on the fly
        xsplot,data,model,wvl,1,150     ;to change color of model curve
        xsplot,data,model,data3=data3,mod3=mod3 ;overplot another set
        xsplot,data,model,mod0=mod0,mod1=mod1   ;overplot alternate models
        xsplot,data,model,mod9=mod9,lcol9=5     ;alt model with specific color
        xsplot,counts,model,keV,/perbin ;plot counts/bin rather than counts/keV
        xsplot,data,model,/ratio        ;plot data/model in lower panel
        xsplot,data,model,/delchi       ;plot delta chi in lower panel
        xsplot,data,model,/resid        ;plot data-model in lower panel
        xsplot,data,model,/fracres      ;plot (data-model)/model in lower panel
        xsplot,data,model,wvl,/xlog,/ylog,xrange=xr,yrange=yr,zrange=zr

        x=findgen(100)+1 & y=50*(sin(x/10)+1.1) & d=0*y & plot,x,y
        for i=0,99 do d[i]=randomu(seed,poisson=y[i]) & xsplot,d,y,x

subroutines
        STAMPLE [SETSYSVAL, LEGALVAR()]
        MID2BOUND()
        REGROUP()

history
        vinay kashyap (Oct09)
```

B.285 ZION2FILENAME [/external]

```
PROJECT:  CHIANTI

      CHIANTI is an Atomic Database Package for Spectroscopic Diagnostics of
      Astrophysical Plasmas. It is a collaborative project involving the Naval
      Research Laboratory (USA), the University of Florence (Italy), the
      University of Cambridge and the Rutherford Appleton Laboratory (UK).


NAME:
       ZION2FILENAME

PURPOSE:

       help locate CHIANTI database files

CATEGORY:

       database.

CALLING SEQUENCE:

      ZION2FILENAME, Iz, Ion, Filename


INPUTS:

       Iz:  nuclear charge of ion of interest, i.e. 26 for Fe
      Ion:   charge state of ion of interest, i.e. 2 for Fe II

 KEYWORDS:

      diel:  set if excitation of this ion is by dielectronic
               recombination

OUTPUTS:

       Filename:  the complete filename and path specification for generic
                 CHIANTI database file, i.e. '.elvlc' suffix is not included


RESTRICTIONS:
```

```
       !xuvtop must be set


EXAMPLE:

            > zion2filename,26,2,filename
            > print,filename
            > /data1/xuv/fe/fe_2/fe_2   assuming !xuvtop = '/data1/xuv'

MODIFICATION HISTORY:
       Written by:     Ken Dere
       March 1996:     Version 2.0
      Sept  1996:     Modified for use with VMS
      December 1998:  Modified to diel keyword

      V.5, 29-May-2002, Giulio Del Zanna (GDZ)
                  generalized directory concatenation to work for
                  Unix, Windows  and VMS. Added keyword name to output just
                  the name of the file and changed the dielectronic  keyword.

VERSION     :   5, 29-May-2002
```

B.286 ZION2SYMB [/util]

```
procedure       zion2symb
                convert atomic number and ionic state to spectroscopic symbol

syntax
        zion2symb,z,ion,symbol,ziform=ziform

parameters
        z       [INPUT; required] atomic number
        ion     [INPUT] ionic state (0=undefined, 1=neutral, 2=singly
                ionized, etc.)
        symbol  [OUTPUT] string containing element designation
                (e.g., "he_2", "He II", "HeII", "he 2", "he", "He",
                depending on ZIFORM)

keywords
        ziform  [INPUT] string containing "Z" and "ION" to describe the
                output format:-
                  if "Z" is capitalized, symbol as is (e.g., "He", not "he").
                  if "z", symbol in small letters (e.g., "fe", "ca", etc).
                  if "ion" is in lower case, ion in arabic numerals.
                  if "ION" or "Ion", ion in roman numerals.
                  symbol and ion separated by whatever "Z" and "Ion" are
                    separated by.
                * default is "ZION"
        oform   [INPUT] kept here only for backwards-compatibility
                * WARNING -- will be phased out >>very<< soon
        _extra  [JUNK] here only to prevent crashing the system

restrictions
        cannot return the "+-" format (e.g., "O^+5")

requires
        INICON

history
        vinay kashyap (Nov98)
        OFORM bug correction (VK; Apr99)
        changed call to INITSTUFF to INICON (VK; 99May)
        allowed OFORM to be "Z<sep>number" and "Z<sep>ion" (VK; MMJul)
        changed keyword OFORM to ZIFORM; converted to IDL5 notation (VK; AugMM)
```