User guide to Spectroscopy reduction V.5

Contents

User guide to Spectroscopy reduction

1. Initial CCD Image Processing

First step is to examine a flatfield using **implot** to determine the overscan and trim sections of the chip. When using **implot** you can use the "e" key on the keyboard to expand around sections of the plot to get a magnified view to better help determine the trim and overscan. These sections will be entered in **ccdproc** under the *biassec:* and *trimsec:* respectfully (if the trim section is already given then just put "image" in *ccdproc*). When entering data use the format ex: {1:1024,1:1024}; meaning in this example we want to keep all sections of the chip in the X and Y. **Note:** in Iraf the trim section is the section of the chip that contains good data **NOT** the sections that you want to get read of.

At this point if you have not already sorted your files or if the camera didn't already do this for you you should. Sorting the files by *imagetyp* is useful when running task that you only want to run on certain images. You can do this by using a command line called **hedit.**

Example: :hedit bias*.imh imagetyp zero add+

This will add the line *imagetyp: zero* in the image header of all files named biasxxx.imh. To check and see if this has worked you can use **imhead** to look at the image header. Using *imhead bias*.imh l+* | *page* will list the long header of each biasxxx.imh image by page. After doing this for your bias frames you can go ahead and do this for the flat, dark,object and sky images now by replacing *zero* with *flat, dark,object and sky* respectfully.

Now that the image type has been added to all of your images you can create directories for each type to keep the reduction process easier. You will want to create the directories Bias, Flat, Dark and a subdirectory in Flat called Sky. To do this use the command **mkdir**.

Example: :mkdir Bias

Note: For spectroscopy you want to set the *dispaxis* = 1 in all the image headers by using **hedit**.

1.2 Bias Images

Now that you have sorted your files you are ready to combine your flat, dark and bias images. We will start with the bias frames using the task **zerocombine.** To check the parameters of **zerocombine** type in *epar zerocombine,* this allows you to edit the parameters. The figure 1 as a useful guide to the parameters of *zerocombine*. When you are ready to return to Iraf type "*Ctrl d"* to save the parameters and get back to the cl: prompt or you can type *:go* to run the task. Before we run the task first lets move all the Bias frames into the Bias directory by typing *imrename @bias Bias/.* Now cd into the Bias directory and create an input file by using the command line **files** as in the example. This will create a input file that contains all the images in the folder.

Example: :files *.fits >> input

Now that we have an input file we need to create an output file from this file with the changes 20030311.15.01.fits to zero.01.fits. To do this emacs input and type Ctrl Space to mark the upper left corner then goto the bottom right and type Ctrl x rt to replace the box with zero. Now type Ctrl x Ctrl w to save as output. Now *imstat *.fits > value* to create a value file with the image statistics. In this file you want to get red of everything but the mean values. After changing type Ctrl x Ctrl s Ctrl x Ctrl c. After this is done we are ready to use **imarith** to to divide all the bias images and create the output images by the example.

Example: :imarith @input / @value @output

Now we can use the **zerocombine** task to create your superbias.fits image that will be used to do the zero correction on the rest of the images. Use the figure below as a guide for the parameters.

Figure 1:

```
PACKAGE = ccdred
   TASK = zerocombine

input   =              zero*  List of zero level images to combine
(output =      superbias.fits) Output zero level name
(combine=              median) Type of combine operation
(reject =           crreject) Type of rejection
(ccdtype=                  ) CCD image type to combine
(process=                 no) Process images before combining?
(delete =                 no) Delete input images after combining?
(clobber=                 no) Clobber existing output image?
(scale  =               none) Image scaling
(statsec=                  ) Image section for computing statistics
(nlow   =                  0) minmax: Number of low pixels to reject
(nhigh  =                  1) minmax: Number of high pixels to reject
(nkeep  =                  1) Minimum to keep (pos) or maximum to reject (neg)
(mclip  =                yes) Use median in sigma clipping algorithms?
(lsigma =                 3.) Lower sigma clipping factor
(hsigma =                 2.) Upper sigma clipping factor
(rdnoise=           GTRON11) ccdclip: CCD readout noise (electrons)
```

```
(gain   =          GTGAIN11) ccdclip: CCD gain (electrons/DN)
(snoise =                0.) ccdclip: Sensitivity noise (fraction)
(pclip  =              -0.5) pclip: Percentile clipping parameter
(blank  =                0.) Value if there are no pixels
(mode   =                 ql)
```

This will create a combined bias named superbias.fits in your image directory. Now *imcopy* the superbias.fits back to the main directory. First epar **ccdproc** to edit the parameters. At this point we want to fix any bad pixels with a pix file. Create a file listing all your bad pixels and call it something like badpix.dat.

Example of badpix.dat file: 149 149 1 800 (bad pix between column 149 and 149 and line 1 to 800)

Now in the parameters of **ccdproc** you can specify to run bias (image superbias.fits), fill in the *trimsec* and *biassec* and take out the bad pixels (file badpix.dat). We do not at this time want to correct for flat fielding. Use figure 2 as a guide to the parameters of **ccdproc**. Notice that the only things set to yes are the overscan, trim and zero! Then run **ccdproc** in the main directory.

Figure 2:

```
PACKAGE = ccdred
   TASK = ccdproc

images  =             *.fits  List of CCD images to correct
(output =                  ) List of output CCD images
(ccdtype=                  ) CCD image type to correct
(max_cac=                 10) Maximum image caching memory (in Mbytes)
(noproc =                 no) List processing steps only?
(fixpix =                 no) Fix bad CCD lines and columns?
(oversca=                yes) Apply overscan strip correction?
(trim   =                yes) Trim the image?
(zerocor=                yes) Apply zero level correction?
(darkcor=                 no) Apply dark count correction?
(flatcor=                 no) Apply flat field correction?
(illumco=                 no) Apply illumination correction?
(fringec=                 no) Apply fringe correction?
(readcor=                 no) Convert zero level image to readout correction?
(scancor=                 no) Convert flat field image to scan correction?
(readaxi=               line) Read out axis (column|line)
(fixfile=                  ) File describing the bad lines and columns
(biassec=              image) Overscan strip image section
(trimsec=              image) Trim data section
(zero   =     superbias.fits) Zero level calibration image
(dark   =     superdark.fits) Dark count calibration image
(flat   =    superflat.fits) Flat field images
(illum  =     supersky.fits) Illumination correction images
(fringe =                  ) Fringe correction images
(minrepl=                1.) Minimum flat field value
(scantyp=          shortscan) Scan type (shortscan|longscan)
(nscan  =                  1) Number of short scan lines
```

```
(interac=              yes) Fit overscan interactively?
(functio=          legendre) Fitting function
(order  =                1) Number of polynomial terms or spline pieces
(sample =                *) Sample points to fit
(naverag=                1) Number of sample points to combine
(niterat=                0) Number of rejection iterations
(low_rej=                3.) Low sigma rejection factor
(high_re=                3.) High sigma rejection factor
```

1.3 Dark Images

1.4 Flat Images

Now you are ready to normalize your flat field images. We start out by doing the same thing as for the Bias images. Use *imrename @flat Flat/* to move the flat images to the flat directory. As before you want to create an input file and use it to create an output file with the same changes as before except this time replace zero with flat. As before make a value file keeping only the mean value column. Use *imarith* to divide the images and **flatcombine** to normalize the flats. use figure 3 as a guide to the parameters of **flatcombine**. After running **flatcombine** on the flat images run the task **response** to create your superflat.fits file. use the figure 4 as a guide to the parameters of this task. Imcopy the superflat.fits into the main directory adn now that this is done you are ready to use **ccdproc** to process the images in your main directory. Be sure to *epar **ccdproc*** and set flat to yes.

Figure 3:

```
PACKAGE = ccdred
   TASK = flatcombine

input   =             flat.*  List of flat field images to combine
(output =              flat) Output flat field root name
(combine=           median) Type of combine operation
(reject =         crreject) Type of rejection
(ccdtype=                 ) CCD image type to combine
(process=               no) Process images before combining?
(subsets=              yes) Combine images by subset parameter?
(delete =               no) Delete input images after combining?
(clobber=               no) Clobber existing output image?
(scale  =             mean) Image scaling
(statsec=                 ) Image section for computing statistics
(nlow   =                1) minmax: Number of low pixels to reject
(nhigh  =                1) minmax: Number of high pixels to reject
(nkeep  =                1) Minimum to keep (pos) or maximum to reject (neg)
(mclip  =              yes) Use median in sigma clipping algorithms?
(lsigma =                3.) Lower sigma clipping factor
(hsigma =                3.) Upper sigma clipping factor
(rdnoise=         GTRON11) ccdclip: CCD readout noise (electrons)
(gain   =         GTGAIN11) ccdclip: CCD gain (electrons/DN)
(snoise =               0.) ccdclip: Sensitivity noise (fraction)
(pclip  =             -0.5) pclip: Percentile clipping parameter
(blank  =               1.) Value if there are no pixels
(mode   =                ql)
```

Figure 4:

```
PACKAGE = longslit
   TASK = response

calibrat=               flat  Longslit calibration images
normaliz=               flat  Normalization spectrum images
response=          superflat  Response function images
(interac=               yes) Fit normalization spectrum interactively?
(thresho=             INDEF) Response threshold
(sample =                 *) Sample of points to use in fit
(naverag=                 1) Number of points in sample averaging
(functio=           spline1) Fitting function
(order  =                50) Order of fitting function
(low_rej=                3.) Low rejection in sigma of fit
(high_re=                3.) High rejection in sigma of fit
(niterat=                0) Number of rejection iterations
(grow   =                0.) Rejection growing radius
(graphic=           stdgraph) Graphics output device
(cursor =                 ) Graphics cursor input
(mode   =                 ql)
```

1.5 Sky Images

Sky flats are easier to reduce. First we need to *imrename @sky to Flat/Sky/* and cd into the directory. epar combine and use figure 5 as a guide then run the task. After that epar illum and use figure 6 as a guide and run this. This will create your supersky.fits that you can use in the main directory to process the rest of the images. Be sure to set sky to yes in **ccdproc**.

Figure 5:

```
PACKAGE = ccdred
   TASK = combine

input   =             *.fits  List of images to combine
output  =                sky  List of output images
(plfile =                 ) List of output pixel list files (optional)
(sigma  =                 ) List of sigma images (optional)
(ccdtype=                 ) CCD image type to combine (optional)
(subsets=                no) Combine images by subset parameter?
(delete =                no) Delete input images after combining?
(clobber=                no) Clobber existing output image?
(combine=           average) Type of combine operation
(reject =          crreject) Type of rejection
(project=                no) Project highest dimension of input images?
(outtype=              real) Output image pixel datatype
(offsets=              none) Input image offsets
(masktyp=              none) Mask type
```

```
(maskval=              0.) Mask value
(blank  =              1.) Value if there are no pixels
(scale  =           mode) Image scaling
(zero   =           none) Image zero point offset
(weight =           mode) Image weights
(statsec=                 ) Image section for computing statistics
(lthresh=          INDEF) Lower threshold
(hthresh=          INDEF) Upper threshold
(nlow   =              0) minmax: Number of low pixels to reject
(nhigh  =              0) minmax: Number of high pixels to reject
(nkeep  =              1) Minimum to keep (pos) or maximum to reject (neg)
(mclip  =           yes) Use median in sigma clipping algorithms?
(lsigma =              3.) Lower sigma clipping factor
(hsigma =              3.) Upper sigma clipping factor
(rdnoise=        GTRON11) ccdclip: CCD readout noise (electrons)
(gain   =       GTGAIN11) ccdclip: CCD gain (electrons/DN)
(snoise =              0.) ccdclip: Sensitivity noise (fraction)
(sigscal=            0.1) Tolerance for sigma clipping scaling corrections
(pclip  =           -0.5) pclip: Percentile clipping parameter
(grow   =              0) Radius (pixels) for 1D neighbor rejection
(mode   =             ql)
```

Figure 6:

```
PACKAGE = longslit
   TASK = illumination

images  =               sky  Longslit calibration images
illumina=         supersky  Illumination function images
(interac=              yes) Interactive illumination fitting?
(bins   =                ) Dispersion bins
(nbins  =                5) Number of dispersion bins when bins = ""
(sample =                *) Sample of points to use in fit
(naverag=                1) Number of points in sample averaging
(functio=          spline1) Fitting function
(order  =               20) Order of fitting function
(low_rej=               3.) Low rejection in sigma of fit
(high_re=               3.) High rejection in sigma of fit
(niterat=               1) Number of rejection iterations
(grow   =               0.) Rejection growing radius
(interpo=           poly3) Interpolation type
(graphic=          stdgraph) Graphics output device
(cursor =                ) Graphics cursor input
(mode   =               ql)
```

2. Background and extracting apertures

For each line or column in the input images a function is fit to the columns or lines specified by the sample parameter. This function is then subtracted from the entire line or column to create an output line or column. This is our last step before we begin to actually extract the spectra from the images and get actual spectra.

1.2 using Background

To start *cd* into the main directory with all of your data. Emacs the object file and add an "a" at the end of all the image names and save this file as output. When running this task a separate window will appear showing your peak for your star. You want to check three different columns to look for cosmic rays and other stars in your field. If the fit is above the background of the spectra you can use the command *:sample xxx:xxx, xxx:xxx* after which you will need to hit "f" for fit and "r" for redraw to reset the sample so that nothing is in the range to bring up the fit. As you go through the images make a mental note of multiple stars so that you can correctly identify your star in the extraction task. Use figure 7 as a useful guide to the parameters for **background.**

Figure 7

```
PACKAGE = longslit
   TASK = background

input   =            @object  Input images to be background subtracted
output  =            @output  Output background subtracted images
(axis   =                2) Axis along which background is fit and subtracte
(interac=              yes) Set fitting parameters interactively?
(sample =     125:145,195:215) Sample of points to use in fit
(naverag=                1) Number of points in sample averaging
(functio=          legendre) Fitting function
(order  =                1) Order of fitting function
(low_rej=                0.) Low rejection in sigma of fit
(high_re=                0.) High rejection in sigma of fit
(niterat=                1) Number of rejection iterations
(grow   =                0.) Rejection growing radius
(graphic=          stdgraph) Graphics output device
(cursor =                 ) Graphics cursor input
(mode   =                ql)
```

1.3 extracting apertures

To start we will need to *epar apsum.* Will use this task to extract one dimensional sums across our apertures. Use figure 8 as a useful guide to the parameters of *apsum.* When running this task a separate window will appear with the aperture and a box identifying the aperture. This should be done for you automatically, but if it does not or if the wrong peak is being identified then you can use "d" to delete and "m" to mark a new aperture. Another important command to remember is *:low and : upper,* these commands can be used to change the lower and upper limits of the box. After you are satisfied with this hit "q" and the fit will appear. Make sure that the fit is a good fit along the line and delete points that are off the fit by hitting "d" to delete and "f" to refit.

Figure 8

```
PACKAGE = apextract
   TASK = apsum

input   =            @output  List of input images
```

```
(output =       ) List of output spectra
(apertur=             ) Apertures
(format =       onedspec) Extracted spectra format
(referen=         ) List of aperture reference images
(profile=             ) List of aperture profile images

(interac=           yes) Run task interactively?
(find   =           yes) Find apertures?
(recente=           yes) Recenter apertures?
(resize =           yes) Resize apertures?
(edit   =           yes) Edit apertures?
(trace  =           yes) Trace apertures?
(fittrac=           yes) Fit the traced points interactively?
(extract=           yes) Extract apertures?
(extras =            no) Extract sky, sigma, etc.?
(review =           yes) Review extractions?

(line   =        INDEF) Dispersion line
(nsum   =           10) Number of dispersion lines to sum or median

(backgro=          none) Background to subtract (none|average|fit)
(mode   =            ql)
```

After *apsum* is run on the data images we need to run *apsum* again on the comparison lamps. To do this we need to first begin by typing *emacs other input.* This will bring up an emacs window with other on top and input on the bottom. What we want to do is make an He lamp image for every data image. If this is done correctly then at the end we will have just as many lines in the other file as in the input file. Now use figure 9 as a guide to the parameters of *apsum* for the Comp Lamps.

Figure 9

```
PACKAGE = apextract
  TASK = apsum

input   =         @other  List of input images
(output =    comp_//@input) List of output spectra
(apertur=             ) Apertures
(format =       onedspec) Extracted spectra format
(referen=         @input) List of aperture reference images
(profile=             ) List of aperture profile images

(interac=            no) Run task interactively?
(find   =            no) Find apertures?
```

(recente= no) Recenter apertures?
(resize = no) Resize apertures?
(edit = no) Edit apertures?
(trace = no) Trace apertures?
(fittrac= no) Fit the traced points interactively?
(extract= yes) Extract apertures?
(extras = no) Extract sky, sigma, etc.?
(review = no) Review extractions?

(line = INDEF) Dispersion line
(nsum = 10) Number of dispersion lines to sum or median

(backgro= none) Background to subtract (none|average|fit)
(mode = ql)

3. Identifying wavelength features

Now we can start to identify wavelength features within our spectra and use this to identify other features using the *identify* and *reidentify* task.

3.2 running identify

Start by typing *epar identify* and using figure 10 as a useful guide to the parameters. The first thing you want to do after running the task is to identify known features in the spectrum. To do this use the "m" key to mark a peak and then type in the known wavelength. Do this for about four different peaks and then hit the "l" key to identify others using the database. Now hit the "f" key to fit this. You will notice that the fit has a distinct shape to it and that there will be outlaying points on this fit for which you can delete by hitting the "d" key and the "f" to refit. after this is done hit "q" to go back to the spectrum and hit "l" to search the database again. Keep doing this till you have about 25 or so peaks identified and a good fit.

Figure 10

PACKAGE = longslit
TASK = identify

images = comp_20050131.15.052a.0001.fits Images containing features to be iden(section=
middle line) Section to apply to two dimensional images
(databas= database) Database in which to record feature data
(coordli= linelists$idhenear.dat) User coordinate list
(units =) Coordinate units

(nsum = 20) Number of lines/columns/bands to sum in 2D image(match = -3.)
Coordinate list matching limit
(maxfeat= 50) Maximum number of features for automatic identif(zwidth = 100.)
Zoom graph width in user units
(ftype = emission) Feature type
(fwidth = 4.) Feature width in pixels
(cradius= 5.) Centering radius in pixels
(thresho= 5000.) Feature threshold for centering
(minsep = 2.) Minimum pixel separation
(functio= spline3) Coordinate function
(order = 3) Order of coordinate function
(sample = *) Coordinate sample regions
(niterat= 0) Rejection iterations
(low_rej= 3.) Lower rejection sigma
(high_re= 3.) Upper rejection sigma
(grow = 0.) Rejection growing radius
(autowri= no) Automatically write to database
(graphic= stdgraph) Graphics output device
(cursor =) Graphics cursor input
crval = 9122 Approximate coordinate (at reference pixel)
cdelt = 1 Approximate dispersion
(aidpars=) Automatic identification algorithm parameters
(mode = ql)

After this task is ran you can run *reidentify* to do the rest of the images. Figure 11 shows the parameters of *reidentify*.

Figure 11

PACKAGE = longslit
TASK = reidentify

referenc= comp_20050131.15.052a.0001.fits Reference image
images = comp_* Images to be reidentified
(interac= no) Interactive fitting?
(section= middle line) Section to apply to two dimensional images
(newaps = yes) Reidentify apertures in images not in reference?(overrid= no)
Override previous solutions?
(refit = yes) Refit coordinate function?

(trace = yes) Trace reference image?
(step = 10) Step in lines/columns/bands for tracing an image(nsum = 10)
Number of lines/columns/bands to sum
(shift = 0.) Shift to add to reference features (INDEF to sea(search = 0.) Search
radius
(nlost = 0) Maximum number of features which may be lost

(cradius= 5.) Centering radius
(thresho= 2000.) Feature threshold for centering
(addfeat= no) Add features from a line list?
(coordli= linelists$idhenear.dat) User coordinate list
(match = -3.) Coordinate list matching limit
(maxfeat= 50) Maximum number of features for automatic identif(minsep = 2.)
Minimum pixel separation

(databas= database) Database

```
(logfile=            logfile) List of log files
(plotfil=                  ) Plot file for residuals
(verbose=                no) Verbose output?
(mode   =                ql)
```

After this is complete we can now save our extracted files to an extracted folder. To do this follow these steps. Make a list file by typing *files 2003*.0001.fits >> list* this will create a file called list with all the image names in it. Now we need to add a REFSPEC to the header by typing *hedit @list REFSPEC1 '("comp_" // $I)' add+ update+ ver-*. Once this is complete run the task *dispcor* on the list images and then type *imrename *.o.fits Extracted/.* Use figure 12 as a guide for the parameters of *dispcor*.

Figure 12

```
PACKAGE = onedspec
   TASK = dispcor

input   =             @list  List of input spectra
output  =         @list // .o  List of output spectra
(lineari=               yes) Linearize (interpolate) spectra?
(databas=          database) Dispersion solution database
(table  =                ) Wavelength table for apertures
(w1     =            INDEF) Starting wavelength
(w2     =            INDEF) Ending wavelength
(dw     =            INDEF) Wavelength interval per pixel
(nw     =            INDEF) Number of output pixels
(log    =               no) Logarithmic wavelength scale?
(flux   =              yes) Conserve flux?
(blank  =                0.) Output value of points not in input
(samedis=                no) Same dispersion in all apertures?
(global =               no) Apply global defaults?
(ignorea=                no) Ignore apertures?
(confirm=                no) Confirm dispersion coordinates?
(listonl=               no) List the dispersion coordinates only?
(verbose=                yes) Print linear dispersion assignments?
(logfile=                ) Log file
(mode   =                ql)
```

4. Cleaning up the spectra

In this section we will discuss the cleanup and final steps of the reduction.

4.2 creating sensitivity functions

In the Extracted folder do a *ccdlist *.o.fits* to determine what flux standards you have from that night. NOTE: also check the log if there is no obvious standard. We want to use the task **standard** to create a std file with sensitivity measurements in it to be used with *sensfunc. Epar standard* and use figure 13 below as a guide to the parameters. Be sure to change the output from std1 to std2 for each different standard

used.

Figure 13

```
PACKAGE = longslit
   TASK = standard

input   =      star.fits Input image file root name
output  =                std  Output flux file (used by SENSFUNC)
(samesta=                  yes) Same star in all apertures?
(beam_sw=                   no) Beam switch spectra?
(apertur=                    ) Aperture selection list
(bandwid=                INDEF) Bandpass widths
(bandsep=                INDEF) Bandpass separation
(fnuzero=  3.6800000000000E-20) Absolute flux zero point
(extinct= onedstds$ctioextinct.dat) Extinction file
(caldir = /usr/local/iraf/iraf/noao/lib/onedstds/spec16cal/) Directory containin
(observa=                 ctio) Observatory for data
(interac=                 yes) Graphic interaction to define new bandpasses
(graphic=              stdgraph) Graphics output device
(cursor =                    ) Graphics cursor input
star_nam=                hr9087  Star name in calibration list
answer  =                  yes  (no|yes|NO|YES|NO!|YES!)
(mode   =                   ql)
```

Now we will use **sensfunc** to create a sensitivity file for each standard frame. *Epar sensfunc* and use figure 14 below as a guide to the parameters. Make sure that you do this for each std file and make the changes sens1 to sens2 accordingly. This will create a sens.0001.fits file for each std file containing sensitivity information. after this is complete we need to *implot* each sens.0001.fits file and find the value at pixel 600 of the image. To do this *implot sens.0001.fits* and press the space bar at the x value of 600. This will output at the bottom of the image the value at that point. Write this down cause we will use this next to divide the original image by this value to normalize each image. Now that we have this value we can use **imarith** to divide the image. To do this type *imarith sens.0001.fits / #value sens.0001.fits* this will divide the image by the value and output to the same file. After this is done to all the images we can now *implot* all the sensitivity functions and *overplot* to get them in the same window to decide which ones to keep. To *overplot* in *implot* just type "o" then ":i filename" and last enter and "l" to *overplot*. Once you have determined the good images use the corresponding std files to create one good std file with all the information in it from the other files. Then run *sensfunc* again, but this time the input is std (the compiled file) and the output is sens. this will create a sensitivity function called sens.0001.fits that we will use to calibrate the rest of the images.

Figure 14

```
PACKAGE = longslit
   TASK = sensfunc

standard=                  std  Input standard star data file (from STANDARD)
sensitiv=                 sens  Output root sensitivity function imagename
(apertur=                    ) Aperture selection list
```

(ignorea= no) Ignore apertures and make one sensitivity functi
(logfile= logfile) Output log for statistics information
(extinct= onedstds$ctioextinct.dat) Extinction file
(newexti= extinct.dat) Output revised extinction file
(observa= ctio) Observatory of data
(functio= spline3) Fitting function
(order = 6) Order of fit
(interac= yes) Determine sensitivity function interactively?
(graphs = sr) Graphs per frame
(marks = plus cross box) Data mark types (marks deleted added)
(colors = 2 1 3 4) Colors (lines marks deleted added)
(cursor =) Graphics cursor input
(device = stdgraph) Graphics output device
answer = yes (no|yes|NO|YES)
(mode = al)

4.3 running calibrate

we will use **calibrate** to apply extinction and flux calibrations to all the images. First we need to *files *.o.fits > input* to create an input file. Next *ccdlist *.o.fits > output* to create an output file that we will edit next. to begin *emacs output* so that we can create an output file that will be used in calibrate. In this file we want to create a list of files that look like the example below.

Example: Processed/gj001.1.fits
Processed/gj001.2.fits

Keep this format up till you have done this for every star in the file and then *ctrl x ctrl s ctrl x ctrl c* to save and exit. Make sure that you have created a Processed folder in the Extracted directory for these files to be created. Once all this is complete you can run **calibrate** on the images. use figure 15 below as a guide to the parameters for this task. Run calibrate in each Extracted folder for each night to create processed images for all nights.

Figure 15
PACKAGE = longslit
TASK = calibrate

input = @input Input spectra to calibrate
output = @output Output calibrated spectra
(extinct= yes) Apply extinction correction?
(flux = yes) Apply flux calibration?
(extinct= onedstds$ctioextinct.dat) Extinction file
(observa= ctio) Observatory of observation
(ignorea= no) Ignore aperture numbers in flux calibration?
(sensiti= sens) Image root name for sensitivity spectra
(fnu = no) Create spectra having units of FNU?
(mode = al)

4.3 clean up

Now we are prepared to clean up the final spectra. *ccd* to the Processed folder to begin this process. *implot* each star to find differences in the spectra. Look for spikes (cosmic rays) or any other differences that need to be clipped out. once you have done this use *splot* to clip out the bad columns/cosmic rays. Once in the *splot* window we can use "we" and "e" to window around a feature. To clip this feature out use "x" at one end and "x" again at the other to draw a line clipping out the bad feature. Once this is done hit "r" to redraw and "i" to save. You can save to the same file and overwrite at this point. After this is done use *sarith* to add the resulting images together. If there is more than two images then you can create a TEMP image that will be used to add to the third or fourth image.

Example: *sarith gj001.1.fits + gj001.2.fits gj001*

If there is three or more files then change the gj001 output to TEMP and use TEMP to add to the third image creating a gj001 output file. This will create a gj001.001.fits file that will be used to create a statistics file. Once this is done type *listpix gj001.001.fits > 2003105.15.gj001*
This will create a 20031005.15.gj001 file that contains all the pixel values for that image. use the format yeardate.telescopesize.star as the output.