University of California
Lick Observatory Technical Report
No. 74

Introduction to Echelle Data Reduction Using the Image Reduction Analysis Facility

Emphasizing The Hamilton Echelle Spectrograph

Christopher W. Churchill

Santa Cruz, California
February 1995

PREFACE

As anyone who has written a manual can attest, it is a never–ending exercise in incompleteness and dissatisfaction. One normally finds that the simplistic model of the world one is committing to permanent form evolves away on too rapid a time scale. Both the Hamilton spectrograph and the Image Reduction and Analysis Facility (IRAF) have been no exception to this phenomenon.

In the time that this manual has been written and re–written, the Hamilton Spectrograph received a new corrector plate (December 1994), which improved its small scale point spread function dramatically. In fact, the severity of the adjacent order overlap addressed often throughout this manual, has been noticeably reduced. Additionally, IRAF V2.10.3 has been released. This manual assumes V2.10.2 throughout, including several "bug" work–a–rounds that are no longer issues in V2.10.3. These "bugs" are outlined in a separate Appendix.

This manual is designed for the IRAF beginner and those who have little experience in reducing CCD data of any kind. The first two chapters serve as an introduction to both these topics and are organized somewhat loosely. The remaining chapters are organized in reduction step order, and provide both a general discussion on IRAFing and specific concerns about the Hamilton Spectrograph. Thus, it is hoped that both beginner and seasoned IRAF users may find this manual useful. A PostScript version of this manual is available via the Internet:

URL=http://gardiner.ucolick.org/~cwc/hamilton/irafman/manual.html

Thanks go to Elizabeth Barker, Ruth Peterson, Matthew Shetrone, and my advisor Steven Vogt for reading and commenting on earlier versions of this manual. Their comments have been very valuable and many find their way into several discussions. Special thanks to Steve Allen for the many hours of his time I consumed with data reduction questions in general and for his very thorough comments on early versions of this manual. Special thanks (again) to my advisor Steven Vogt for all those great things for which great advisors are known. Special thanks to Tony Misch ("You fellas better have some beans") for his excellent advice and fun companionship during those special nights at the Lick 3–meter. Thanks also to Diana Lazo–Foote for her diligent proof–reading and assistance in the publication of this Report.

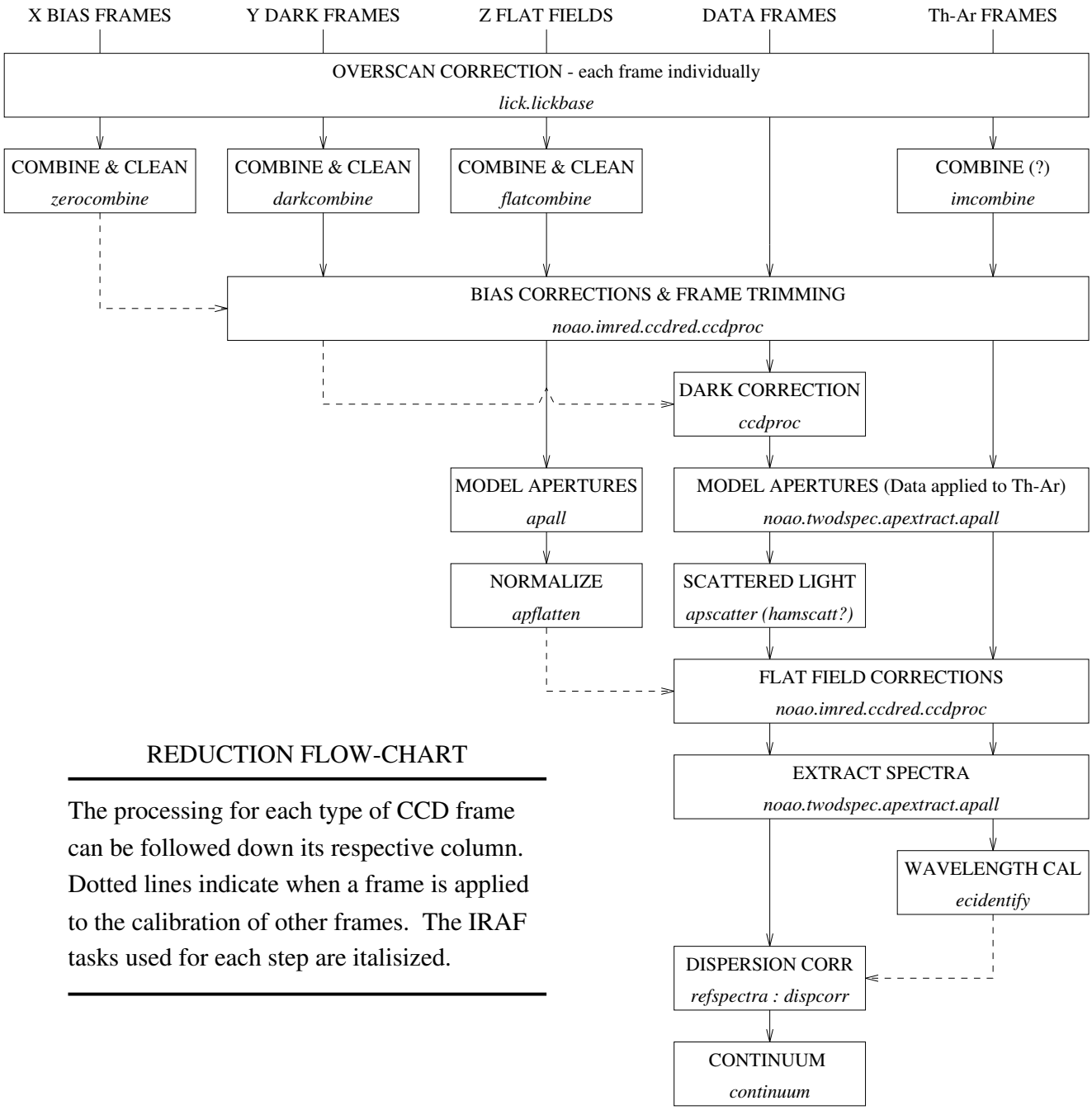
In particular, I express my heartfelt gratitude to J.L. Nelson for his great wisdom and influential inspiration, which will never ever be forgotten.

TABLE OF CONTENTS

PAGE REFERENCE TO FIGURES

PAGE REFERENCE TO TASKS

The following references IRAF tasks and packages to the pages upon which they appear. Tasks are listed in alphabetical order, though they are proceeded by their package locations. Bold face pages give the page upon which the task parameters are listed.



REDUCTION FLOW-CHART

The processing for each type of CCD frame can be followed down its respective column. Dotted lines indicate when a frame is applied to the calibration of other frames. The IRAF tasks used for each step are italisized.

1. INTRODUCTION

This manual has been written with the basic assumption that the reader is not proficient in echelle data reduction nor with the use of IRAF in general. Thus, many of the discussions are a mixture of IRAF tutorial and echelle data reduction specific concerns. Additionally, this introductory section includes a brief discussion of general CCD reduction steps, particularly those related to calibrating electronic artifacts in CCD data frames.

Since the Hamilton Spectrograph (hereafter "the Hamilton") has its own set of "quirks" (don't all spectrographs?), many of the discussions focus on handling these specific issues. So, in principle this manual could be used to learn IRAF reduction for any echelle spectrograph, though in practice one may become over–burdened by certain sections.

It is assumed that the reader is using a UNIX environment and some window system capable of running IRAF, including a Tektronics or gterm plotting window, and some image display software (ie. SAOimtool or XImtool).

1.1. IRAF Background

If one has used IRAF before, one may skip this section with no loss of continuity. IRAF is separated into "packages" and "tasks". Tasks actually perform operations upon the data. Packages are groupings of tasks that are generally related to one another by the operations they perform, such as plotting or general mathematical operations.

If one has never used IRAF, the best way to learn is to have somebody sit down with you and provide a tutorial of fundamental skills: how to fire–up IRAF, display and plot images, move around the environment accessing the on–line help files, and edit the task parameter files.

To access a task, one must have "loaded" its parent package, which is accomplished by simply typing the the package name and hitting [CR]. Each task has an accompanying parameter list, which can be accessed via an "eparam" file. Though a task can be executed directly by typing its name, it is safest to edit the epar file and directly execute by typing ":g", for "go". To edit an epar file, type "epar taskname". To exit an epar file without executing the task, type ":q", which will save the current settings. To exit without saving (modifying) the parameter list, type ":q!".

1.2. Before Using the Hamilton

The most informative source of information describing the Hamilton is the Lick Observatory Technical Report (LOTR) No. 58 by Tony Misch (1991). One may obtain any LOTR by (1) calling the Lick Publications office (408) 459–2201, (2) electronic mail to `publof@lick.ucolick.org`, (3) electronic FAX 408–454–9863 c/o Computer Resources, or (4) telephone FAX 408–426–3115.

Basic information about the Hamilton can be obtained via the Internet from the UCO/LICK

Gopher server (URL=gopher://ucogo.ucsc.edu/). Once hooked up to the UCO/LICK server, click on the options *Observing Information*, *Mt. Hamilton Observing Information*, *Information about Instruments*, and then *Hamilton Echelle Spectrograph*. From the top entry level, one may select the *Mt. Hamilton Gopher Server*, from which several options are available, including *CCD Detector Laboratory Data*, and *Dewar–data*. The last two are valuable for learning the specifications of the currently available instruments.

Several reduction steps involve the computation of some sort of noise model, usually Poisson fluctuations and read–noise in quadrature. Poisson statistics need to be computed in units of electrons (e^-) detected. Two quantities that are important to know are the instrument read–noise (in units of e^-) and the gain factor g.

1.3. CCD Calibration and Reduction Background

Here is presented a brief, yet comprehensive overview of calibration steps which must be performed before the information in the CCD can be considered free of instrumental artifacts. Specific issues pertaining to the LICK Data Acquisition System (LDAS) are addressed. A small emphasis is placed on the "error budget", which one should conscientiously consider when quantifying physical quantities. Following this overview of calibration concerns, a mathematical description of the calibration steps are presented.

For the following, the term "data frame" will be taken to represent the image on the CCD. The errors common to data frames are either multiplicative or additive. If they are multiplicative, one corrects for the error by a division process. If they are additive, one corrects by subtraction. Common instrumentally introduced artifacts include, but certainly are not limited to: 1) Analog to Digital Conversion (ADU), 2) Read–Noise (RN), 3) Amplifier Bias – Overscan (OS), 4) Electronic Bias (BIAS), 5) Thermal Current (DARK), 6) Pixel Gain Variations (FLAT), 7) Saturation, and 8) Charge Transfer Efficiency (CTE).

1.3.1. Analog to Digital Conversion Errors

There are two distinct items to consider due to analogue to digital conversion. (1) For each pixel, the read–out amplifier must convert a measured amount of charge (analogue process) into an integer number of e^- and then store that number as a multi–bit binary code (digitization process). Both a read–noise and a discretization error arise from this process. The latter error is usually negligible compared to the read–noise. For LDAS, the discretization error is $\pm 0.5 e^-$, resulting from integer roundoff. If it is to be budgeted, one usually considers discretization as a contribution to the "effective" read–noise (see below). (2) The data values for each pixel output by a CCD system are in units called DN (digital number) or ADU (Analogue to Digital Units). These two terms are commonly used interchangeably. The gain factor

$$g = \frac{\text{number of } e^- \text{ detected}}{\text{DN}},$$

gives the conversion of DN to e^-. The number of e^- is the physical quantity being measured (proportional to incident photons) and not the DN. Never make signal–to–noise (S/N) calculations in units of DN – always convert to e^- using the gain factor. Each pixel is virtually a separate "detector", having its own gain. However, in general an "average" gain factor for the CCD is quoted and used for the purposes of the noise budget.

1.3.2. Read–Noise

This is simply a noise source that must be included in the error budget. The read–noise is associated with the process of measuring the amount of charge in a given pixel by time integrating the measured current (analogue process). Quoted in number of e^-, read–noise RN is considered a mean quantity for the CCD. In general, one may wish to include discretization error added in quadrature to quote an "effective read–noise". If one co–adds data frames, the read–noise adds in quadrature. If one is co–averaging N frames (co–adding and dividing by the number of frames co–added) and using the average signal value of these frames, then the read–noise is reduced by the factor $\sqrt{N}$.

For LDAS, the read–noise is always greater than the average gain factor, so that the error due to read–noise is resolved. For example, if $g = 10\ e^-/\mathrm{DN}$ and $RN = 5e^-$, then the RN would not be resolved in DN (10 e^- would be required to register 1 DN, and there would be no way to distinguish the contribution of the RN to this digital number). Furthermore, the LDAS quoted RN does not include the discretization error.

1.3.3. Amplifier Bias Level: The Overscan Correction

The amplifier bias is not a noise source, but rather is a measure of the electronic "zero" level that physically indicates zero photons counted. One should think of this as a pedestal to be removed from the data to assure that they are restored to the number of e^- actually *detected* by the CCD. Usually, a region of the data frame is dedicated to storing this bias level. For LDAS, it is a single "overscan" column. One should determine the mean level of the overscan column and subtract this value from all pixels. If a gradient is present across rows, then a linear function may be used to model the overscan data.

1.3.4. Electronic Bias

This is an additive noise source that arises from the simple fact that it takes time to read the CCD. During read–out, certain voltage generating/decrementing processes may systematically introduce structure across the data frame. Note that this is quite different from the amplifier bias. Often much confusion arises from loose usage of the term "bias". Electronic bias should be readily noticeable in so called "bias frames" ($t = 0$ integrations). With LDAS, one must take a $t = 1$ integration with the shutter closed (dark) as an approximation to the bias frame. To increase the S/N at such low counts, one should co–average many individual bias frames to obtain a "calibration bias" frame.

One should always convince oneself that this calibration frame should really be subtracted from the data frame, that the read–out process truly does produce a bias pattern on the data frames. Inspect the calibration frame for *no* bias: (1) the mean pixel count (in DN) should be zero (following overscan correction!), (2) the count distribution function f should be Gaussian with width ($FWHM = 2.35\sigma$), given by

$$f(y_i) = \frac{1}{\sigma\sqrt{2\pi}} e^{-y_i^2/2\sigma} \qquad \sigma = \left\{ \frac{\sum^{N_p} y_i^2}{N_p - 1} \right\}^{1/2} = \frac{\text{read–out noise}}{\sqrt{\text{number of co–added frames}}},$$

where $y_i = g \cdot \mathrm{DN}_i$ is the number of e^- measured in pixel i, and (3) a histogram of number of pixels verses count level should provide a visual "verification" of the distribution, width, and possible skew of the bias calibration data.

With LDAS, the read–out amplifier introduces a 60 Hz pick–up with an amplitude of about 4–8 DN peak to peak (see Fig. 5–1), though this amplitude is not entirely stable throughout the night. LDAS offers two read–out speeds. It is best to use the "slow" read–out option, for the read–out is synchronized with the 60 Hz pick–up in "slow" mode. This results in a repeatable "corrugated" surface over data frames.

1.3.5. Thermal (Dark) Current

This is additive noise that arises from either one or both of two sources: (1) a general background level resulting from thermally generated e^- that is quite low (few e^-/hr), (2) locally "hot" pixels that display vastly higher rates of thermal e^- generation. For both, the accumulation rate of e^- is proportional to exposure time and sensitive to temperature ($\propto e^{-kT}$). "Super–hot" pixels (those which saturate on the time scale of the read–out process) will create "hot" columns, since the pixel values downstream in that column pass through the hot pixel during the read process. Theoretically, dark current is linear with time. The treatment for dark current is to take *many* data frames with the shutter closed with integration times appropriate to one's program. A "calibration" dark frame is then created by co–averaging and cosmic ray removal.

There are a number of philosophies regarding dark current and the construction of one or several dark calibration frames with different integration times. One can assume time linearity and either (1) scale a single calibration dark frame to the data frame exposure time, or (2) interpolate between several calibration dark frames to the exposure time of the data frame before subtracting off this scaled calibration dark frame. Or, one can attempt to avoid the assumption of time linearity and obtain dark calibration frames with integration times the same as the data frame exposure times. The main concern is whether dark current subtraction should be really performed, since this process invariably leads to a reduction of S/N.

1.3.6. Pixel Gain Variations

Gain variation from pixel to pixel is a multiplicative noise source which is due to (1) differences in λ dependent quantum efficiencies (QE = efficiency of conversion of incident photons into e^-) from pixel to pixel or across the CCD, (2) fringing problems, which are λ dependent interference creating global interference patterns (a serious yet subtle problem), (3) dust particles and the like in the optical path that cause discernible and repeatable patterns on the data frame. A fourth problem may be quantum efficiency hysteresis, where the QE of a pixel depends upon its exposure history. The technique of removing these "gain" variations involves generating a calibration flat field frame in an attempt to scale the gain in each pixel to the "average" gain of the CCD.

Making a calibration flat field frame is highly application dependent. If one is attempting to remove pixel to pixel variations, one usually divides the data frame by a normalized calibration flat field (mean of 1 DN), where the difference between the value in a pixel and unity gives the percent variation from the mean gain. Since flat fielding is of great importance and is quite unique in its application to echelle data reduction, further discussion is reserved for §1.6.2 & §10.

1.3.7. Saturation

This is a situation when information is totally lost. There are two types of saturation: (1) the

A/D converter, which converts number of counted e^- into a binary representation of DN, has a maximum value limited by the number of bits used to store the value,

$$\mathrm{DN}_{max} = 2^{(\#\text{ of bits})} - 1.$$

For LDAS, the number of bits is 15. If the A/D converter saturates it will reach a constant maximum output value. What one sees in the image is not the usual $2^{15} - 1 = 32767$, but roughly 29300 DN, since the amplifier bias level (as determined from the overscan column) is subtracted out on–line. (2) The "well depth", or the total number of e^- that a single pixel can store for later read–out has a maximum. When a pixel reaches this maximum, no more incident photons can be stored in *that* pixel (QE = 0), though e^- are still liberated by the photons. If this case, the excess liberated charge from the saturated pixel may spill over to adjacent pixels up and down the column, since the potential barriers are smallest in that direction. If the pixels are being saturated at a very high rate then enough charge may spill out to flow between columns too. The result is a streaming along rows and columns, and the degradation of nearby signal.

LDAS is designed so that the A/D converter saturates before the CCD wells saturate (because the gain is up high to properly sample the read–noise), meaning that the CCD full well is much larger than the A/D converter maximum count capacity. This results in the added benefit that one remains within the linear response range of the CCD (meaning the gain is independent of the number of detected e^-) all the way until A/D saturation occurs.

1.3.8. Charge Transfer Efficiency

The charge transfer efficiency is the fraction of e^- that are successfully passed onto the next position during the horizontal and vertical shifting associated with the read–out process. Many CCDs have CTE in the 0.99995 (!) range; still, this can leave up to 10% of the charge in "downstream" pixels of a 2048x2048 CCD. Bad CTE results in minor smearing with the smears pointing away from the read–out edge of the CCD. If the CTE is low for a given row, or set of pixels, the smearing effect can be quite devastating. Though corrections may be applied, it is safest to keep the science away from these pixels.

1.4. Basic Calibration Reduction Steps

The following outline is a simple demonstration of the removal of the most common instrumental artifacts. Assuming one has a satisfactory data set (good CTE, no saturation, also see §1.6) with several un–calibrated data frames [RAW], N_B bias frames [BIAS], N_D dark frames [DARK], and N_F flat field frames [FLAT], a simple series of additive and multiplicative calibration steps are to be performed.

The desired "real" data frame ([DATA]) is convolved with instrumental "information" according to

$$[\mathrm{RAW}] = \Big\{[\mathrm{DATA}] + [\mathrm{scatter}]\Big\}[\mathrm{fringe}][\mathrm{flat}] + [\mathrm{dark}] + [\mathrm{bias}] + \mathrm{OS}_{\mathrm{RAW}}$$

where $\mathrm{OS}_{\mathrm{RAW}}$ is the overscan of [RAW], the frame [FLAT] is the convolution of the transmission function and quantum efficiency, and the [scatter] and [fringe] are higher order effects that will need removal, but are not discussed here. One wishes to obtain [DATA] while minimizing the errors due to additive and multiplicative processing. Normally, one assumes that the instrument and electronics are stable with time and then beats down the noise in the [BIAS], [DARK], and [FLAT] frames by

averaging them with "cleaning" algorithms according to

$$\langle \mathrm{BIAS} \rangle = \frac{1}{N_B} \sum_i^{N_b} \left([\mathrm{BIAS}]_i - \mathrm{OS}_i \right) ,$$

$$\langle \mathrm{DARK} \rangle = \frac{1}{N_D} \left\{ \sum_i^{N_d} \left([\mathrm{DARK}]_i - \mathrm{OS}_i \right) \right\} - \langle \mathrm{BIAS} \rangle ,$$

$$\langle \mathrm{FLAT} \rangle = \frac{1}{N_F} \left\{ \sum_i^{N_f} \left([\mathrm{FLAT}]_i - \mathrm{OS}_i \right) \right\} - \langle \mathrm{BIAS} \rangle - \langle \mathrm{DARK} \rangle .$$

Then one performs the operation

$$\{ [\mathrm{DATA}] + [\mathrm{scatter}] \} \, [\mathrm{fringe}] = \frac{([\mathrm{RAW}] - \mathrm{OS}_{\mathrm{RAW}}) - \langle \mathrm{BIAS} \rangle - \langle \mathrm{DARK} \rangle}{\langle \mathrm{FLAT} \rangle} .$$

These processes will be the topics of §3, §5, §6, §7, and §11.

1.5. Overall IRAF Reduction Step Outline

The following flow chart illustrates the echelle data reduction steps followed by this manual. The right hand column lists the tasks that are used to perform the step.

Reduction Step	IRAF package(s).task
copy FITS images to disk, convert to IRAF format	fits2disk (UNIXscript); dataio.t2d dataio.rfits
overscan correct all frames	lick.lickbase
setup translation table insert 'IMAGETYP' cards	noao.imred.ccdred.setinstrument images.hedit
clean and combine bias frames clean and combine dark frames clean and combine flat frames	noao.imred.ccdred.zerocombine noao.imred.ccdred.darkcombine noao.imred.ccdred.flatcombine
trimming, bias and dark subtraction from data, arcs, and flat field	noao.imred.ccdred.ccdproc
model aperture locations, sizes	noao.twodspec.apextract.apall
normalize flat field	noao.twodspec.apextract.apflatten
apply flat field correction to data and arcs	noao.imred.ccdred.ccdproc
remove scattered light	noao.twodspec.apextract.apscatter
extract spectra	noao.twodspect.apextract.apall
wavelength calibrate attach dispersion solution apply dispersion solution	noao.imred.echelle.ecidentify noao.imred.echelle.refspectra noao.imred.echelle.dispcor
normalize continuum	noao.imred.echelle.continuum

1.6. Example Data Set

Given the above discussion of instrumental calibration concerns, one sees the need for a series of bias, dark, and flat fielding frames. Further, one will eventually convert the extracted spectra to a wavelength scale. This requires an image of a Thorium–Argon (Th–Ar) Lamp, or some other bright emission arc for which lines are identified.

Assumed for this basic outline are that, at a minimum, the reader has a data set on an Exabyte tape comprised of:

2–3	Th–Ar Lamps	wavelength arcs
X	Bias frames	$t = 1$ sec Darks
Y	Dark frames	
Z	Quartz Lamp	flat field frames
1	Quartz Lamp	mapping orders
1	Water Star	fast rotator
N	Program Data	

where X, Y and Z are absolutely no less than 3. The Quartz Lamp and "water" star frames will be used to model the aperture locations and sizes for the flat field and the program data, respectively. Additionally, the water star may be used to remove telluric features, and to some extent, fringing that occurs red–ward of 7500Å.

For this example reduction run, we will use a data set consisting of 2 Th–Ar Lamps, 5 Bias frames, 5 Dark frames, 5 Flat frames, 2 "order mapping" frames (1 Quartz Lamp and 1 water star), and 6 Data frames. For clarity, the following names for the frames will be (see §2 for how to name data frames:)

ARCS	BIAS	DARKS	FLATS	MISC	DATA
thar01	bias01	dark01	flat01	quartz	data01
thar02	bias02	dark02	flat02	H2Ostr	data02
	bias03	dark03	flat03		data03
	bias04	dark04	flat04		data04
	bias05	dark05	flat05		data05
					data06

1.6.1. Th–Ar Lamps

The Hamilton itself is very stable, having been known to move less than a fraction of a pixel over the course of the night. This movement is likely due to weight change as liquid nitrogen evaporates from the dewar. Therefore, unless one is measuring extremely precise velocity differentials or something of that nature, one need not acquire a Th–Ar Lamp between each observation. Keep in mind that the Th–Ar Lamp is shuttled into the light path in the "slit room"; information about changing light paths due to telescope flexure cannot be obtained using Th–Ar Lamps.

One needs to acquire at least two Th–Ar Lamps, one short exposure ($t = 3$s) and one long exposure ($t = 10$s) with a bg38 filter. The latter exposure is vital to gleaning the few Thorium lines

in the red, which are swamped out by the highly saturated Argon lines. Wavelength calibration in the red will be quite unconstrained if one does not obtain this "red burn" Th–Ar Lamp.

A Th–Ar Atlas, taken under nominal excitation conditions for the Hamilton, is available: LOTR No. 73 by Matthew Shetrone (1994).

1.6.2. Flat Fields

The width of the apertures (cross–dispersion direction) is controlled by the Decker height.* If one obtains flat fields with Decker settings the same as the program data, the S/N at the aperture cross dispersion edges in the normalized calibration flat field will be very low where they align with the program data. Thus, one would be multiplicatively introducing a higher noise level along the aperture edges of the program data by flat fielding with such a frame.

There are two philosophies designed to circumvent this problem. 1) The flat field frame should be obtained with a "wider" Decker setting, at least twice as wide as the program Decker. The aim is to keep the "noisy" flat field data outside the program data apertures (see Fig. 1–1). 2) Open the Decker fully and obtain a global illumination of the CCD. This latter method has the attraction of being very simple to normalize, but does not retain the color sensitivity component to the QE. Thus, the former technique actually gives a more "accurate" measure of the pixel to pixel variations for the program data.

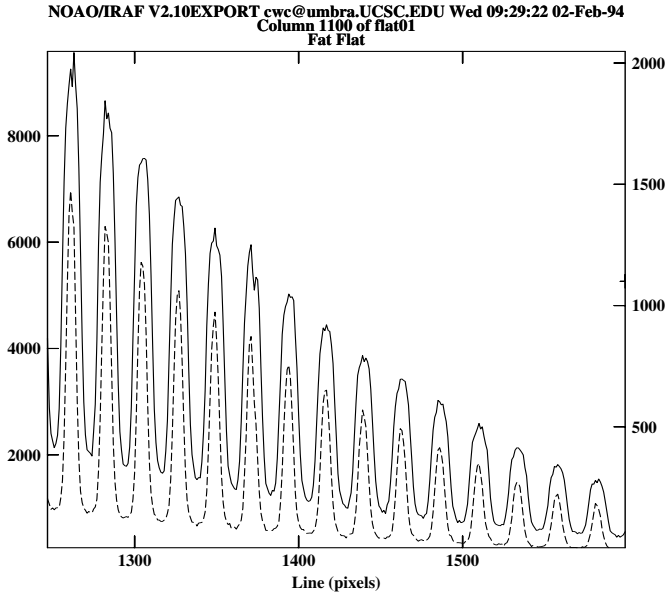


Figure 1–1. A cross section over plot of "wide" Decker flat field apertures (solid lines) and program Decker apertures shows the desired effect of high signal count of the flat field data at all cross dispersion pixels of the program data. This insures that the flat field data signal–to–noise is high in all pertinent pixels while retaining the color sensitivity term.

The following reduction treatment assumes that the flat frames have a "wide" Decker setting, twice that of the program data.

* cf. Misch (1991) LOTR No. 58 for details.

2. DATA I/O: IRAF FORMAT

The Lick Data Acquisition System (LDAS) stores each data frame on tape as standardized FITS formated files. These FITS files need to be "converted" to IRAF format. Here we discuss different methods for transferring one's FITS files from tape to IRAF files on disk.

In the UNIX environment the UNIX, "dd" command may be used. Within IRAF, at least two tasks are useful. From the help file of the **dataio** package, the pertinent tasks are:

```
rfits - Convert a FITS image into an IRAF image
  t2d - Fast tape to disk copy
```

The task **rfits** may be used to convert FITS format to IRAF format directly from tape to disk (see §2.4.2).

2.1. A Word About IRAF Format

IRAF stores three files for each image. The first two are identical copies of the image headers. One is hidden as a "..imagename.imh" file and the second is simply "imagename.imh". These header files live in the image's "home" directory. IRAF creates a "pixel" subdirectory in the home directory, where the actual data values are stored in binary files with names such as "imagename.pix".

Never perform UNIX file manipulations on these files. All image and header file manipulations should be performed in the IRAF environment, since IRAF will know to do the proper bookkeeping on all related files.

2.2. Setting the UNIX 'TAPE' Environment

The first step to down–loading one's data is to properly configure one's UNIX environment. If the tape drive is public, one will need to "request" the drive to obtain sole ownership (if it is private, omit this step). Second, one must set the UNIX TAPE environment variable to point to the drive requested.

For example, if one wishes to use the drive "nrst0" (no–rewind standard tape 0), then (in UNIX) type "request /dev/nrst0" (it is suggested that one always use the no–rewind option). Then, to set the environment variable, type "setenv TAPE /dev/nrst0". The environment variable must be set in the parent window of the IRAF session. For more information a "man–page" for "request" is on–line. When done with the drive, type "release /dev/nrst0" as a courtesy to other users.

Having loaded the tape, one can examine its status by typing "mt status". The UNIX "mt" commands are useful for skipping forward quickly, rewinding, and ejecting the tape.

2.3. Copying FITS files from Tape to Disk

This section is for those who are more comfortable examining their data in their original FITS format prior to converting them with IRAF. If one wishes to by–pass this step, the method described in §2.4.2 is a more expedient approach to obtaining IRAF format data.

2.3.1. The UNIX Approach

The UNIX "dd" method is quite sluggish. UCSC graduate students Steve Allen and Jesus Gonzalez have written a very nice UNIX script called "fits2disk". To read the FITS format data from Exabyte tape to the current directory using the UNIX script "fits2disk" type something like "fits2disk data 0 26", which pulls the files 0 through 26 off tape and places them in the current directory with the names "data.0", "data.1", etc. For LDAS, file "0" is a tape directory, basically all FITS headers concatenated into one large header.

2.3.2. The t2d IRAF Approach

The task **t2d** is very simple, though not much faster than the "fits2disk" UNIX script. Before reading from tape, one must **allocate** the tape drive. Task **allocate** resides in the **system** package. One must thus know the IRAF *device* designation, which is specific to one's local computing facility. These should be tabulated in the file "dev$devices". In IRAF, one may see the locally available devices by typing "type dev$devices". At UCSC, Exabyte drives are designated "mtc".

Allocate the tape drive by typing "allocate mtc". To check the device status, type "devstatus mtc", which will output something like:

```
# Magtape unit mtc status Tue 14:15:03 01-Sep-93 user cwc
file = -1
record = -1
nfiles = 0
tapeused = 0 Kb
pflags = 0
```

Now, epar into **t2d** to set the parameters:

```
PACKAGE = dataio
   TASK = t2d
input   =                  mtc  Input file descriptor
ofroot  =               ut1002  Output file root name
files   =                16,30  List of files
(verbose=                  yes) Print out progress reports
(errigno=                  yes) Assume an error record is zero bytes long
(mode   =                   ql)
```

The param "input" is the drive designation. The param "ofroot" is the root name of all files written to disk. In this example, it is the UT date of the data set. The param "files" are the file numbers to read from tape. Files will be read in ascending order, regardless of the order of the list. Reading will terminate when EOT is reached, thus a list such as "1–999" may be used to read all the files on the tape.

Remember, the first file LDAS stores is a tape directory. Task **t2d** starts counting from 1, so file 1 is the LDAS directory! This example will result in data log entries 15 and 29 being read from tape and saved to disk as "ut100216" and "ut100230". Upon completion, type "deallocate mtc" to **deallocate**, or release the device.

2.4. Converting FITS to IRAF

The conversion of FITS format to IRAF format requires the task **rfits** from the **dataio** package. If one copied one's FITS files directly to disk, one may simply convert them to IRAF format. However, **rfits** is capable of copying FITS files from tape, converting them directly to IRAF format, and writing them to disk.

The single main concern during the conversion process is "scaling" of the data. If the FITS header cards BSCALE and BZERO are set to 1 and 0 respectively, then no scaling is needed – otherwise the data have been scaled by the data aquisition system. The scaling follows

$$\mathrm{DN} = [\mathrm{bscale}] \times \mathrm{DN}^{'} + \mathrm{bzero}$$

where DN is the output value, $\mathrm{DN}^{'}$ is the recorded FITS value, and "bscale" and "bzero" are the numerical values of the BSCALE and BZERO cards, respectively. LDAS sets BSCALE='1' and BZERO='-64'.

2.4.1. Disk to Disk

Assuming one has followed one of the methods for copying FITS files directly to disk, one may convert all FITS images to IRAF images using **rfits**. An easy way to do this in assembly line mode is to use an input file list (called an "@file") with the FITS file names and an output file list with the IRAF image names. The task **files** in the package **system** is useful for creating such @files. In the current directory with the FITS images, type "files data.* > fitsfiles" to create an input file list called "fitsfiles". Create a parallel file (call it "iraflist", for example) with corresponding IRAF image names. The order of the file names appearing in "iraflist" must be in one to one correspondence with those in "fitsfiles". If one wishes to store the IRAF images in different directories one may include the paths, so IRAF knows where to store them. Or, one may simply wish to blow away the FITS images after the images have been successfully converted. In the package **dataio**, epar into **rfits** and set the parameters:

```
PACKAGE = dataio
   TASK = rfits
fits_fil=           @fitsfiles  FITS data source
file_lis=                       File list
iraf_fil=            @iraflist  IRAF filename
(make_im=                  yes) Create an IRAF image?
(long_he=                   no) Print FITS header cards?
(short_h=                  yes) Print short header?
(datatyp=                    r) IRAF data type
(blank  =                   0.) Blank value
(scale  =                  yes) Scale the data?
(oldiraf=                   no) Use old IRAF name in place of iraf_file?
(offset =                    0) Tape file offset
(mode   =                   ql)
```

The param (scale = yes) is set because LDAS adds 64 DN to the data to avoid storing values

less than zero. This zero offset is recorded in the header card BZERO = '−64.' This scaling restores the data to their proper values. The param (datatyp = r) is set to store the data as type "real". See the help file for options here. The drawback with real type data is that it doubles the required disk space; on the otherhand, certain IRAF tasks require real type data during processing, and one would need to do the proper bookkeeping of converting the data type as necessary.

2.4.2. Tape to Disk

This method has the advantages of economy of number of steps and disk space. It is by far the most direct approach. When one has reached the level of mass production, one will actually find this method to be advantageous.

In **rfits**, the param "fits_file" is either an @file list of disk files or a tape device specification ("mtc"), the latter of which will read the files specified by the "file_list" parameter from tape to disk. Read both §2.3.2 and §2.4.1 regarding allocating the tape drive and setting the parameters of **rfits**. After allocating the device set the **rfits** parameters as above, but with the following exceptions:

```
fits_fil=                  mtc  FITS data source
file_lis=                2-999  File list
iraf_fil=              ut1002  IRAF filename
```

This example copies all files (not including the LDAS "0th" file) from device "mtc" and gives them the root name "ut1002". As an example of the "iraf_fil" parameter, reading files 2 and 3 from a FITS tape with a value of (iraf_fil = data) will produce the files "data0002" and "data0003", whereas reading the same two files with a value of (iraf_fil = data1,data2) will produce the files "data1" and "data2".

As the task executes, the header scrolls by for each frame when it is read from the tape. Files can be read in any order, for example (file_lis = 2–5,10–15) and (iraf_fil = calframe) will result in the IRAF files "calframe0002" through "calframe0005" and "calframe0010" through "calframe0015", the 1st through 4th and 9th through 14th FITS *data files* from tape.

3. OVERSCAN: AMPLIFIER ZERO POINT

The overscan correction is simple removal of read amplifier offset. During the read–out of each row, after the last real column is registered, the amplifier voltage level is read (zero signal voltage level) and this value is inserted as the last column. As a zeroth order overscan correction, LDAS *subtracts this overscan value from the row prior to data storage.* This is a poor treatment of removing the zero level of the amplifiers, since it is noisy from row to row. The task **lickbase** in the **lick** package, accounts for pre–subtraction of the overscan column. It restores each row by adding back the overscan level and then invokes the **colbias** task to allow interactive polynomial fitting of the overscan column.

Additionally, **lickbase** provides the option of inserting the RDNOISE (read–noise in e^-), GAIN (conversion of DN to e^-), and DISPAXIS (direction of spectral dispersion on CCD) header cards into the image headers. Also inserted is the TRIMSEC card, which defines the area of the data frame that is to be fully reduced. The TRIMSEC card, as inserted by **lickbase**, defines the entire data frame minus the overscan column. For a 2048 column image, it is sometimes necessary to trim the image by a few more columns at the peripheries due to spurious signal. One may wish to modify the hard–wired value of the TRIMSEC card (see §8.1).

As before, it is easiest to create an @file for processing a data set. Use the "iraflist" file created for **rfits** as the input list and create a file (call it "proclist", for example) for the output list. Use the task **sections** in the **images** package. Typing the command "sections @iraflist//.bl > proclist" will append ".bl" to the file names and store them in file "proclist".

> Caution: **lickbase** requires disk space for its calculations. Be sure that the "tmp" directory is on a spacious disk. A safe capacity is at least three times the size of a single IRAF image. For reference, a 1600x2048 data type "real" pixel image consumes 13108 kbytes. A personal tmp directory can be defined prior to executing **lickbase** with a command such as:
>
> set tmp = "/scratch/umbra/cwc/tmp/"
>
> The last "/" is critical when defining path variables. Do not fail to include it.

The default fitting parameters can be set in the **colbias** epar file (**lickbase** is really a cl script that drives **colbias** and **imarith**). If one wishes to see a verbose output of the **imarith** operation, set the (verbose = yes) in the **imarith** parameter file. Leave the (title =) field blank, or all the **lickbase** output files will be retitled! It is suggested that prior to overscan correction, one sets the **colbias** default fitting parameters as follows (partial parameter listing):

```
(median =                   no) Use median instead of average in column bias?
(interac=                  yes) Interactive?
(functio=            legendre) Fitting function
```

```
(order  =                    2) Order of fitting function
(low_rej=                   3.) Low sigma rejection factor
(high_re=                   1.) High sigma rejection factor
(niterat=                    3) Number of rejection iterations
```

Since the overscan correction is simply the removal of a "pedestal", one should fit a low order (order = 2) polynomial (functio = legendre). Do not attempt to model the higher frequency behavior of the amplifier zero point. See below for a discussion on sigma rejection factors. Now, in the **lick** package, epar into **lickbase** and set the parameters:

```
PACKAGE = lick
   TASK = lickbase
input   =            @iraflist  Input images
output  =            @proclist  Output images
(rdnoise=                  4.7) Readout Noise   (if >= 0)
(gain   =                 1.33) Gain            (if >  0)
(dispaxi=                    1) Dispersion Axis (if >  0)
(observa=                     ) Observatory database key
(ilist  =                     )
(olist  =                     )
(mode   =                   ql)
```

The param (dipsaxi = 1) indicates dispersion along lines (IRAF term for rows), the (rdnoise = 4.7) and (gain = 1.33) params are the read–noise and gain factor for Dewar 13 (Lick3 Orbit 464–8 2048x2048) as of 1994 March–03. These three useful header cards will be used time and time again in later processing steps. Upon typing ":g", one will be prompted by the **colbias** task of the **noao imred bias** package to perform the overscan correction. Upon confirmation, a plotting window will appear (Fig. 3–1). The plot is the overscan column with a functional curve fit using the **colbias** fitting parameters.

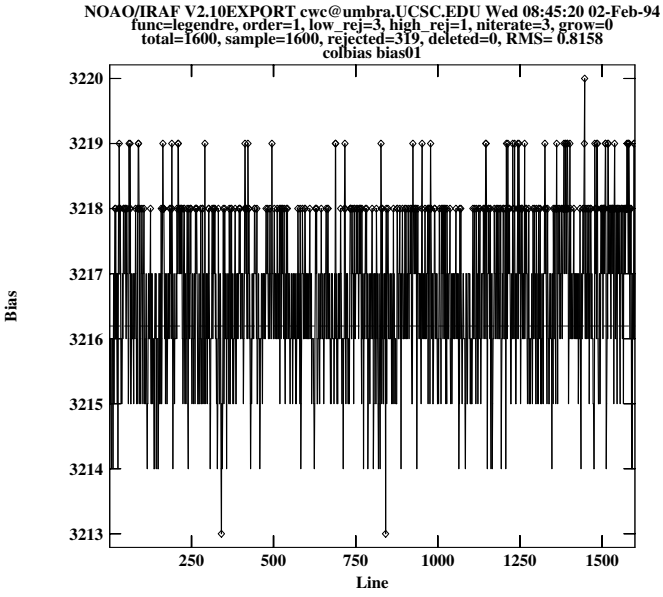


Figure 3–1. The **icfit** (interactive cursor fitting) routine of **colbias** is shown. The plot header displays the fitting parameters used in this example. Points flagged with diamonds have been rejected from the fit, which has been rejection iterated 3 times to best remove outliers based upon the :upper and :lower sigma limits. The overscan column of a bias frame usually has no structure, so that fitting a constant is appropriate (:order = 1).

One is now in the **icfit** procedure of the **colbias** task. Typing "?" will display the available commands. The graph header displays their present settings. Some important colon commands are:

```
:show          Show the values of all the parameters
:func [value]  Fitting function (chebyshev, legendre, spline3, spline1)
:grow [value]  Rejection growing radius
:order [value] Fitting function order
:low [value]   Low rejection threshold
:high [value]  High rejection threshold
:niter [value] Number of rejection iterations
```

Colon commands are not updated until "f" (Fit the data and redraw or overplot) is keyed. Some other useful keystrokes are:

```
?       Print options
d       Delete data point nearest the cursor
f       Fit the data and redraw or overplot
q       Exit the interactive curve fitting.
r       Redraw graph
s       Select a sample range
u       Undelete the deleted point nearest the cursor
```

WARNING: do not throw too many key strokes at **icfit** in rapid succession. If one gets "stuck", the technique to save oneself is to type [CTRL–C] and then slowly repeat the "?" key until the terminal beeps. Then, type "r" for a redraw of the graph.

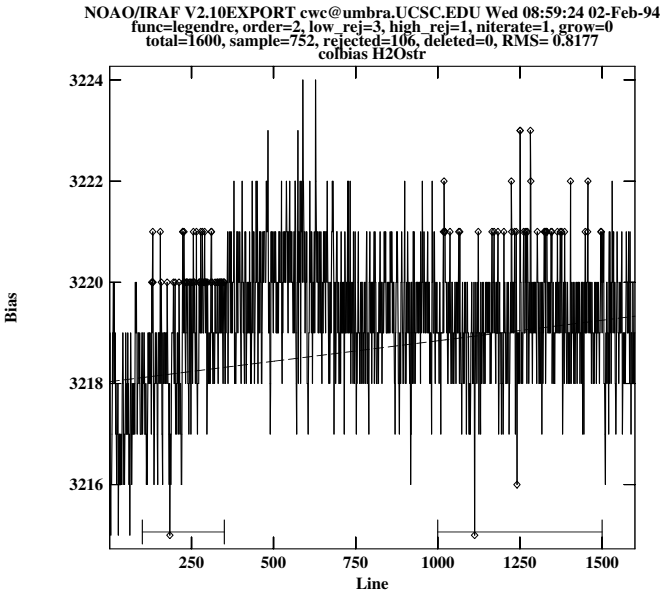


Figure 3–2. The overscan column of a "water star" is seen to have structure that mimics the global illumination along columns. Fit the overscan to lines where the illumination is low, for the high signal biases the overscan column too high due to a slow RC decay in the electronics. The horizontal bars across the bottom indicate the regions designated by the ":sample" parameter. Outside these regions, the data are ignored in the fit.

Fit a constant or a straight line to the end regions of the overscan signal. This can be accomplished by using a small ":high #" with ":func legendre", and ":order 1" (or 2). The reasoning is that lines

with high signal tend to bias the overscan too high due to a slow read–out amplifier RC constant. *The accurate zero level of the amplifier corresponds to the lines where the illumination of the CCD is low.* Even then, overscan correction tends to be overestimated by a few DN out of roughly 3200 (an average level for low signal frames).

To see the current **icfit** parameter settings, type the colon command ":show", which will output something like:

```
Wed 08:59:21 02-Feb-94
colbias H2Ostr
function = legendre
grow = 0.
naverage = 1
order = 2
low_reject = 3.
high_reject = 1.
niterate = 1
sample =  100:350,1000:1500
```

Note the ":sample" parameter, which one may invoke to force the fitting to preferred lines along the column, as illustrated in Fig. 3–2. The "sample" range can be selected using the "s" key. Place the graphics cursor at the sample start location and type "s" to mark the lower limit. Then move the cursor to the upper limit and key "s" again. A horizontal bar marking the sample range to be included in the fit will appear. One can mark several "sample" ranges.

The header cards inserted by **lickbase** can be viewed using the **imhead** task in the **images** package; for example, typing "imhead bias01.bl" gives the following (partial header listing):

```
. . .
. . .
. . .
BIASSEC = '[2048:2048,1:1600]'
TRIMSEC = '[1:2047,1:1600]'
DATASEC = '[1:2047,1:1600]'
CCDSEC  = '[1:2047,1:1600]'
CCDSUM  = '1 1     '
OVERSCAN= 'LICKBASE Wed 1993 Sep 8 22:46:14 PDT'
RDNOISE =                  4.7
GAIN    =                 1.33
DISPAXIS=                    1
```

When using the **imhead** task, pipe the output into **page** to enforce single page scrolling.

4. TRANSLATION TABLE AND 'IMAGETYP' CARDS

The processing tasks work in an highly automated fashion provided that a system to differentiate calibration frames from data frames has been constructed. In other words, once one sets up key words in the headers of each image that define whether that image is a flat, a bias, or a dark frame, and then provides IRAF with a means to "translate" those key words, one will be able to streamline processing. The IMAGETYP header cards provide the former function, and the "translation table" provides the latter, the translation of the IMAGETYP cards.

4.1. Defining the Translation Table

To set up the table load the packages **noao**, **imred**, and **ccdred**, and type "setinstrument ?". This will result in the output:

```
direct          Current headers for Sun plus CCDPROC setup for direct CCD
specphot        Current headers for Sun plus CCDPROC setup for spectropho-
                  tometry, ie GoldCam, barefoot CCD
foe             Current headers for Sun plus CCDPROC setup for FOE
fibers          Current headers for Sun plus CCDPROC setup for fiber array
coude           Current headers for Sun plus CCDPROC setup for Coude
cyrocam         Current headers for Sun plus CCDPROC setup for Cryo Cam
echelle         Current headers for Sun plus CCDPROC setup for Echelle
kpnoheaders     Current headers with no changes to CCDPROC parameters
fits            Mountain FITS header prior to Aug. 87 (?)
camera          Mountain CAMERA header for IRAF Version 2.6 and earlier
Instrument ID (type q to quit) (echelle):
```

Type "echelle" and hit [CR] for echelle format (or just hit [CR] if it is the default as shown above). One is then piped into the epar file of **ccdred** which looks like:

```
PACKAGE = imred
   TASK = ccdred
(pixelty=            real real) Output and calculation pixel datatypes
(verbose=                 yes) Print log information to the standard output?
(logfile=             logfile) Text log file
(plotfil=                    ) Log metacode plot file
(backup =                    ) Backup directory or prefix
(instrum= ccddb$kpno/echelle.dat) CCD instrument file
(ssfile =             subsets) Subset translation file
(graphic=            stdgraph) Interactive graphics output device
(cursor =                    ) Graphics cursor input
(version=      2: October 1987)
(mode   =                   ql)
```

The above settings suffice. The important param is 'instrum = ccddb$kpno/echelle.dat', which is

the translation table used by the KPNO echelle. This table works fine for LDAS. At this point, type ":q" to quit and save, following which one is piped into the epar file of **ccdproc**. Exit with ":q" it at this time. The translation is now set up.

The parameter value 'ccddb$kpno/echelle.dat' is a text file which contains the translation table, which reads:

```
 DARK            dark
 BIAS            zero
 OBJECT          object
'DOME FLAT'      flat
'PROJECTOR FLAT' flat
'COMPARISON'     comp
'SKY FLAT'       object
```

Now, the IMAGETYP cards must be set. For example, if a certain image has the card "IMAGETYP = flat", **ccdproc** will know that the image is a "DOME FLAT", "PROJECTOR FLAT" or "whatever flat", and not a DARK, BIAS, or OBJECT frame.

4.2. Inserting 'IMAGETYP' Cards

Unfortunately, LDAS does not insert the IMAGETYP header cards in the header. When (or if) this feature is added to LDAS, this step will be eliminated. The task **hedit** in the **images** package is used for editing, inserting, and deleting header cards. Again, it is easiest to create @files to process large data sets. The task **files** in the **system** package can be used to create an @file for each image type. Assuming the naming convention outlined in §1.6, type the following commands:

command to make @file	IMAGETYP value
files bias*.bl > biaslist	'zero'
files dark*.bl > darklist	'dark'
files flat*.bl > flatlist	'flat'
files thar* quar* H2O* data* > objlist	'object'

For example, "files bias*.bl > biaslist" will create a file called "biaslist", with the image names that are to have their IMAGETYP values set to "zero". Epar into **hedit** and set the following (this example is for bias frames):

```
PACKAGE = images
   TASK = hedit
images  =             @biaslist  images to be edited
fields  =              IMAGETYP  fields to be edited
value   =                  zero  value expression
(add    =                   yes) add rather than edit fields
(delete =                    no) delete rather than edit fields
(verify =                   yes) verify each edit operation
(show   =                   yes) print record of each edit operation
(update =                   yes) enable updating of the image header
(mode   =                    ql)
```

Be sure to have spelled IMAGETYP correctly. Since the (fields = IMAGETYP) card is being *added*, the param (add = yes) is set. Setting (add = no) and (delete = no) will instruct **hedit** to *change* the value of a pre–existing header card. Do the same for the remaining filelists, being sure

to set the "value" param as dictated by the translation table corresponding to each @file. When done, one may wish to spot check the card insertions using **imhead** in the **images** package.

5. CALIBRATION BIAS FRAME

LDAS is not capable of $t = 0$ second integrations, thus bias frames are actually $t = 1$ second dark frames. It is wise to visually inspect the bias frames, since these frames reveal "blemishes" that occur during read–out. One may decide that bias correction is not needed for one's program.

LDAS introduces a 60 Hz pick–up during the read–out of the CCD. This pick–up has a frequency of about 450 pixels and a peak to peak of about 6–8 DN. It has been observed to vary as little as 4 DN, depending upon the PG&E hookup. It is clocked in the "slow" read mode.

5.1. Inspecting The Bias Frames

Three methods for inspecting the bias frame(s) are given below. Remember that the raw bias frames are compromised by cosmic ray hits, which manifest as a high–end tail on the distribution of pixel intensities.

5.1.1. Visual Inspection

Visual inspection should be made using the task **implot** in the **plot** package. Typing "implot bias01.bl" will plot the center line in the plotting window. One can plot any column, line, or column or line averages. To see the list of helpful commands type "?". Below are a few key cursor commands:

```
c               plot a column
e               expand plot by marking corners of viewport
l               plot a line
q               quit
r               redraw
<space>         print coordinates and data value
:a N            set number of lines or columns to be averaged
:c M [N]        plot column[s] M [to N]
:i image        open a different image
:o              overplot next vector
:l M [N]        plot line[s] M [to N]
:log+, log-     enable, disable log scaling in Y
:x x1 x2        fix plotting range in X (call with no args to unfix)
:y y1 y2        fix plotting range in Y (call with no args to unfix)
```

For example, to average columns 91 to 110, enter ":a 20" (average 20 columns) and then ":c 100". To expand on a region type "e" on the lower left corner of the region. The prompt "again:" will appear. Move the cursor to the upper right and type "e"again. To return to the full view, type

"wa". Figure 5–1 shows an example, clearly illustrating the 60 Hz pick–up.

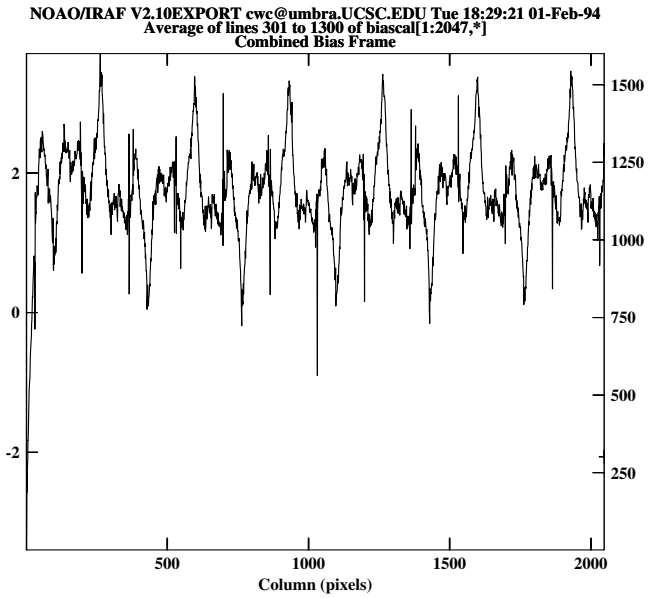


Figure 5–1. LDAS introduces a 60 Hz pick–up, which is clocked to the read–out provided the "slow" option is used. This figure is actually the combined and cleaned calibration bias frame, which has had most spurious features and cosmic rays removed. This cut across columns is an average of 1000 lines, showing "corrugated–like" features down lines. The greater–than–zero mean of the frame indicates that the overscan was slightly under–estimated.

5.1.2. Distribution Check

A second approach to examining bias frames is to view a histogram. "Perfect" bias frames should have a Gaussian distribution with a mean DN level of zero and width equal to the read–noise divided by the gain (= read–noise in DN). As a side note, one should do *all* statistics in units of $e^- = g \cdot \mathrm{DN}$. Since the exponent of a Gaussian is unitless, consistency of units in either DN or e^- will work for bias frame inspection.

The task **imhist** in the **images** package is useful, except that it is a bit tricky to get out a useful looking histogram. Epar into **imhist** and set the params:

```
PACKAGE = images
   TASK = imhistogram
image   =            bias01.bl  Image name
(z1     =                -30.) Minimum histogram intensity
(z2     =                 30.) Maximum histogram intensity
(binwidt=                  5.) Resolution of histogram in intensity units
(nbins  =                 60) Number of bins in histogram
(autosca=                 no) Adjust nbins and z2 for integer data?
(top_clo=                 no) Include z2 in the top bin?
(hist_ty=             normal) Type of histogram
(listout=                 no) List instead of plot histogram?
(plot_ty=               line) Type of vectors to plot
(logy   =                yes) Log scale y-axis?
(device =           stdgraph) output graphics device
(mode   =                 ql)
```

Remember that the output is with a "ylog" scale. This setting resulted in a fairly good normal distribution between ±30 with a frequency peaking near 10^6 at DN = 0. Increasing "z2" will allow one to see the high–end spotty tail (cosmic rays, etc.). Do not over sample by making "nbins" too large. This produces aliasing effects. If one's histogram is a set of jagged peaks, then reset "nbins" to a lower number. To make a list of frequency verses DN values (for use with plotting software) then set (listout = yes) and use a redirect '>' to send it to a file. Performing **imhist** on a cleaned and combined bias frame provides a more realistic view of the LDAS bias.

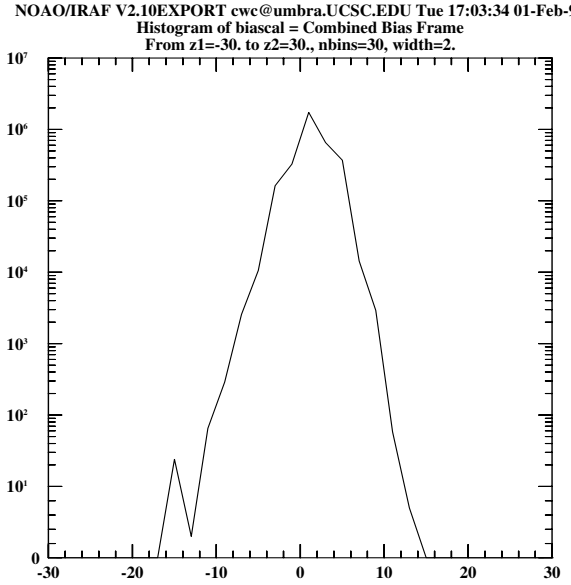


Figure 5–2. A histogram of the combined and cleaned calibration bias frame shows a somewhat symmetric distribution with a mean near zero. For a bias frame that indicates no biasing effects, the width of the distribution should be the system read–noise (here in DN) divided by the square root of the number of combined frames. The features of the calibration frame are probably real, since to survive the combination process any feature had to be statistically present in all frames (more on this later).

5.1.3. Statistical Check

One can run a quick and dirty statistical check on the bias frames using the **imstatistics** task in the **images** package. The default outputs are: number of pixels, mean, standard deviation, minimum value, and maximum value. Higher moments can be useful for checking that the distribution is a normal distribution. To add the fields for skew and kurtosis, epar into **imstat** and set the "fields" parameter:

```
PACKAGE = images
   TASK = imstatistics
images  =            @biaslist  Images
(fields = image,npix,mean,stddev,skew,kurtosis,min,max) Fields to be printed
(lower  =                INDEF) Lower cutoff for pixel values
(upper  =                INDEF) Upper cutoff for pixel values
(binwidt=                  0.1) Bin width of histogram in sigma
(format =                  yes) Format output and print column labels?
(mode   =                   ql)
```

The output will look something like:

```
#   IMAGE      NPIX    MEAN    STDDEV      SKEW  KURTOSIS      MIN       MAX
bias01.bl   3275200   -1.47      4.32     102.8     39625.   -19.11     2077.
bias02.bl   3275200  -1.501     4.592     181.7     97344.   -18.61     3145.
bias03.bl   3275200  -1.461     4.441      166.     99460.   -15.96     3232.
bias04.bl   3275200  -1.369     4.112     99.08     46484.   -19.45     2421.
bias05.bl   3275200  -1.343     3.808     37.79     10846.   -19.79     1541.
```

A `MEAN` less than zero indicates that the overscan correction has likely been overestimated. For these data, the read–noise is 8.8 e^-, or 8.8/2.75=3.2 DN, less than the `STDDEV` of the frames. The distribution is not "normal" as indicated by the skew and kurtosis. The positive skew indicates a high end tail on the distribution, dominated by hot columns and cosmic rays. The positive kurtosis (leptokurtic) indicates that the peak of the distribution is "pointy", and not "rounded" (concavity of the distribution curve is highly positive above and/or below the mean value).

5.2. Cleaning and Combining Bias Frames

The calibration bias frame should be an average or a mean of the individual bias frames with cosmic rays rejected and other spurious positive spikes removed. Assuming that a calibration bias frame is to be constructed, epar into the task **zerocombine** in the **noao imred ccdred** package. This task is a script which uses the tasks **ccdproc** and **imcombine**, but has default parameters specifically set for bias images.

```
PACKAGE = ccdred
   TASK = zerocombine
input   =            @proclist  List of zero level images to combine
(output =              biascal) Output zero level name
(combine=              average) Type of combine operation
(reject =             crreject) Type of rejection
(ccdtype=                 zero) 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) 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=              rdnoise) ccdclip: CCD readout noise (electrons)
(gain   =                 gain) 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)
```

Even though the input includes all overscan corrected files, as listed in @proclist, **zerocombine** will process only the frames with IMAGETYP='zero' header cards! If one has not set up the translation table and inserted the IMAGETYP cards, **zerocombine** will simply not proceed. Here, the output file will be "biascal". The param (combine = average) could also have been set to the "median". See the help file for details. Averaging is computationally quicker. The param "reject" could have

one of many settings:

```
minreject - Reject the minimum value at each pixel
maxreject - Reject the maximum value at each pixel
minmaxrej - Reject both the min. and max. values at each pixel
 crreject - Detect and reject cosmic rays at each pixel
threshold - Reject pixels above and below specified thresholds
  sigclip - Apply a sigma clipping algorithm to each pixel
avsigclip - Apply a sigma clipping algorithm to each pixel
```

Use (reject = crreject) to eliminate cosmic rays and spurious peaks outside the "hsigma" of each pixel. Do not tighten down the param "hsigma", for this may reduce the effectiveness of the rejection routine and actually degrade the resulting statistical distribution of the calibration frame. Details of the rejection methods are outlined in the help file for **imcombine**. Below is the entry for "crreject".

```
CRREJECT
    The output image is the average of the input images, excluding
    cosmic rays.  Cosmic rays are excluded at each pixel by
    rejecting values that exceed the median of the input images by
    more than 'highreject' times sigma at that pixel.  The sigma is
    estimated from a noise model by finding the quadratic sum of
    the photon noise, the read-noise, and a 'scale' noise that
    depends linearly upon the median value.  The parameters for the
    noise model are found in the 'noisepar' pset.  The sigma image
    is the square-root of the variance about that mean, divided by
    the square-root of the number of non-rejected values.  Unless
    the images are at the same exposure level they should be
    scaled.
```

Actually, the "noisepar" pset is overridden by the **zerocombine** params "rdnoise", "gain", and "snoise". The latter is a noise scaling that may be determined in the **noisemodel** task of the **stsdas** package, but is mostly useful only for high count levels.*

The params (rdnoise = rdnoise) and (gain = gain) inform **zerocombine** that these values are in the image header cards RDNOISE and GAIN. The param (mclip = yes) is set so that the estimate of the true "intensity" is from a median rather than from an average with high and low pixel values excluded. The param (pclip = -0.5) is the default setting that selects the fraction of the pixels above the median to use for computing the clipping sigma.

The task **imcombine** may alternatively be used directly and offers greater flexibility, having options such as thresholding, masking, and offsetting. The output of **zerocombine** is:

```
Sep  9 14:28: IMCOMBINE
  combine = average, scale = none, zero = none, weight = none
  reject = crreject, mclip = yes, nkeep = 1
  rdnoise = rdnoise, gain = gain, snoise = 0., hsigma = 3.
  blank = 0.
               Images  Rdnoise   Gain
          bias01.bl.imh     8.8   2.75
          bias02.bl.imh     8.8   2.75
          bias03.bl.imh     8.8   2.75
          bias04.bl.imh     8.8   2.75
```

* If one wishes to study the noise characteristics of the instrument and CCD, the interactive **stsdas.hst_calib.wfpc.noisemodel** task is very useful. At the very least, the help file is an excellent read.

```
        bias05.bl.imh     8.8   2.75
Output image = biascal, ncombine = 5
```

Performing an **imstat** on the image "biascal" should show statistically marked improvement:

```
# IMAGE      NPIX   MEAN    STDDEV      SKEW  KURTOSIS       MIN       MAX
biascal   3275200  -1.45     1.693   -0.1788    0.1024    -11.46     6.795
```

The distribution width (standard deviation) in DN should now be read–noise/gain divided by $\sqrt{\text{ncombine}} = (8.8/2.75)/\sqrt{5} = 1.43$, which is close to the computed `STDDEV`. Use **imhist** to better interpret the **imstat** output. One should find a fairly well behaved distribution (see Fig. 5–2).

6. CALIBRATION DARK FRAME

The dark current calibration frame is an average of many frames with cosmic rays removed minus the calibration bias frame. Since subtraction is commutative, one may create the averaged dark frame and then subtract the calibration bias frame from it to make the calibration dark frame. Often, one assumes time linearity and scales the dark current for exposure time of the object data. Again, one must decide whether to perform a dark current correction on the data. It depends upon one's application.

Similar to the **zerocombine** task for constructing the calibration bias frame is the task **darkcombine**, which is a script for **ccdproc** and **imcombine** with default parameters specific to dark frames. In the **noao imred ccdred** package, epar into **darkcombine** and set the parameters:

```
PACKAGE = ccdred
   TASK = darkcombine
input   =             @proclist  List of dark images to combine
(output =              darkcal) Output dark image root name
(combine=              average) Type of combine operation
(reject =             crreject) Type of rejection
(ccdtype=                 dark) 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  =             exposure) 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 keep (pos) or maximum reject (neg)
(mclip  =                  yes) Use median in sigma clipping algorithms?
(lsigma =                   3.) Lower sigma clipping factor
(hsigma =                   3.) Upper sigma clipping factor
(rdnoise=              rdnoise) ccdclip: CCD readout noise (electrons)
(gain   =                 gain) 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)
```

As with the bias frame, the param (combine = average) is set instead of the "median" option. The param (reject = crreject) is found to be superior to the "avsigclip" option even though the latter is recommended in the **imcombine** help file in the case of "only a few images". The "avsigclip" routine is very slow. The param (scale = exposure) will scale each frame and weight average them according to

$$\text{scale} = \frac{\langle T \rangle}{T},$$

$$\text{weight} = \frac{(NT)^{1/2}}{\sum (NT)^{1/2}},$$

where $\langle T \rangle$ is the average exposure time, T is the exposure time of the frame, and N is the number of frames combined. Setting (scale = exposure) enforces the explicit assumption of time linearity. Other "scale" options exist; see the **darkcombine** and **imcombine** help files. The output of **darkcombine** is:

```
Sep 10 10:25: IMCOMBINE
  combine = average, scale = exposure, zero = none, weight = none
  reject = crreject, mclip = yes, nkeep = 1
  rdnoise = rdnoise, gain = gain, snoise = 0., hsigma = 3.
  blank = 0.
                Images  Rdnoise   Gain
          dark01.bl.imh     8.8   2.75
          dark02.bl.imh     8.8   2.75
          dark03.bl.imh     8.8   2.75
          dark04.bl.imh     8.8   2.75
          dark05.bl.imh     8.8   2.75
  Output image = darkcal, ncombine = 5
```

The "exposure time" of "darkcal" will be $\langle T \rangle$. The output of **imstat** for the dark frames follows:

```
#   IMAGE      NPIX    MEAN    STDDEV      SKEW  KURTOSIS       MIN       MAX
dark01.bl   3275200   -1.08     3.585      151.    69872.    -20.13     2196.
dark02.bl   3275200  -1.018      5.33     548.7   485793.    -16.39     5280.
dark03.bl   3275200  -0.9773     3.93     156.2    59746.    -20.13     2188.
dark04.bl   3275200  -0.9066     3.91     208.8   108994.    -16.24     2500.
dark05.bl   3275200  -1.167     3.409     147.9    80592.    -16.31     2332.
darkcal     3275200  -1.042     1.296   -0.1885    0.3051    -12.03     6.322
```

Note that the `MEAN` of the dark frames is negative (these example darks happen to be short and this is an artifact of over estimating the overscan column). At this point, the frame "darkcal" has not been bias frame corrected. When the calibration frames are processed by **ccdproc**, the dark calibration frame will be bias corrected prior its application as dark correction. For example purposes, the output of **imstat** is shown for the above "darkcal" image following bias correction in **ccdproc**:

```
#   IMAGE      NPIX    MEAN    STDDEV      SKEW  KURTOSIS       MIN       MAX
darkcal     3275200  0.4083     1.212   -0.1163    0.1368    -9.164     6.265
```

6.1. Really Treating Dark Current

Linearly scaling the dark current may not be accurate enough for one's program. Rumor has it that a cubic polynomial was once used to interpolate between dark frames of different exposure lengths. The safest bet is to make the exposure times of the dark frames equal to the exposure times of the program data.

7. CALIBRATION FLAT FIELD FRAME

Flat fields should be combined using the task **flatcombine**, following which this combined flat frame should have the illumination functions removed leaving pixel to pixel sensitivities (more on this later). To combine the flat frames, epar into **flatcombine** and set the parameters:

```
PACKAGE = ccdred
   TASK = flatcombine
input   =            @proclist  List of flat field images to combine
(output =             flatcomb) Output flat field root name
(combine=              average) Type of combine operation
(reject =             crreject) Type of rejection
(ccdtype=                 flat) CCD image type to combine
(process=                   no) Process images before combining?
(subsets=                   no) Combine images by subset parameter?
(delete =                   no) Delete input images after combining?
(clobber=                   no) Clobber existing output image?
(scale  =                 none) 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);maximum to reject (neg)
(mclip  =                  yes) Use median in sigma clipping algorithms?
(lsigma =                  2.5) Lower sigma clipping factor
(hsigma =                  2.5) Upper sigma clipping factor
(rdnoise=              rdnoise) ccdclip: CCD readout noise (electrons)
(gain   =                 gain) 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)
```

The parameters are similar to **zerocombine** and **darkcombine** (see §5.2 about **zerocombine** regarding the parameter set). The **flatcombine** output is presented below:

```
Sep 10 22:39: IMCOMBINE
  combine = average, scale = none, zero = none, weight = none
  reject = crreject, mclip = yes, nkeep = 1
  rdnoise = rdnoise, gain = gain, snoise = 0., hsigma = 2.5
  blank = 1.
                Images  Rdnoise   Gain
          flat01.bl.imh     8.8   2.75
          flat02.bl.imh     8.8   2.75
          flat03.bl.imh     8.8   2.75
          flat04.bl.imh     8.8   2.75
          flat05.bl.imh     8.8   2.75
  Output image = flatcomb, ncombine = 5
```

One may wish to use **display** to give the results a once over.

8. TRIMMING, BIAS AND DARK CORRECTIONS

Trimming off the overscan column (and more?), and correcting the additive artifacts of bias and dark current may be treated with a single execution of the task **ccdproc**.

Prior to performing these processes on all overscan corrected images, edit the @file "proclist" for the images input list, replacing all "bias##.bl", "dark##.bl", and "flat##.bl" frames with "biascal", "darkcal", and "flatcal", respectively. As with **zerocombine**, **darkcombine**, and **flatcombine**, if one has properly set the translation table and correctly inserted the IMAGETYP cards, **ccdproc** will know which frames are applied as calibration frames and which are "object" frames.

8.1. Trimming Concerns

For echelle data, trimming off the overscan column is best performed prior to modeling the apertures and normalizing the flat field calibration frame, particularly if one is reducing only a specific sub–window of one's data frames. One may also wish to look for edge columns or lines that are "bad". At this writing, column 1 classifies as a "hot" column.

Recall that **lickbase** inserted default values for the header cards DATASEC and TRIMSEC. These may or may not be set to the regions of one's choosing. To override the **lickbase** defaults, one may simply define the trim section in the **ccdproc** parameter file. The values of both parameters are set to the section of the data frame to be kept. Be sure to understand the notation when setting the header cards. The convention is:

```
TRIMSEC = [startcol:endcol,startline:endline]
DATASEC = [startcol:endcol,startline:endline]
```

8.2. Running ccdproc

The **ccdproc** processing is outlined in a 16 step algorithm in the help file. This is an excellent, informative, and well written source of CCD processing basics. Two important points are (1) once an image is processed by **ccdproc**, a CCDPROC header card is inserted with a statement of the processes performed, and (2) **ccdproc** intelligently uses this information to streamline further processing; it will not duplicate previous processing. However, for calibration frames (IMAGETYP = 'zero', 'dark', or 'flat'), once a CCDPROC header card is inserted into their headers, **ccdproc** will not perform *any* further processing *on them.*

In the package **noao imred ccdred**, epar into **ccdproc** and set the parameters for trimming, bias correction, and (possibly) dark correction:

```
PACKAGE = ccdred
   TASK = ccdproc
images  =            @proclist  List of CCD images to correct
```

```
(ccdtype=                    ) CCD image type to correct
(max_cac=                   0) Maximum image caching memory (in Mbytes)
(noproc =                  no) List processing steps only?
(fixpix =                  no) Fix bad CCD lines and columns?
(oversca=                  no) Apply overscan strip correction?
(trim   =                 yes) Trim the image?
(zerocor=                 yes) Apply zero level correction?
(darkcor=                 yes) Apply dark count correction?
(flatcor=                  no) Apply flat field correction?
(illumco=                  no) Apply illumination correction?
(fringec=                  no) Apply fringe correction?
(readcor=                 yes) Convert zero level image- readout correction?
(scancor=                  no) Convert flat field image- scan correction?
(readaxi=                line) Read out axis (column|line)
(fixfile=                    ) File describing the bad lines and columns
(biassec=                    ) Overscan strip image section
(trimsec=               image) Trim data section
(zero   =                    ) Zero level calibration image
(dark   =                    ) Dark count calibration image
(flat   =                    ) Flat field images
(illum  =                    ) Illumination correction images
(fringe =                    ) Fringe correction images
(minrepl=              -1000.) Minimum flat field value
(scantyp=          shortscan) Scan type (shortscan|longscan)
(nscan  =                  1) Number of short scan lines
(interac=                 no) Fit overscan interactively?
(functio=          chebyshev) 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=                  1) Number of rejection iterations
(low_rej=                 3.) Low sigma rejection factor
(high_re=                 3.) High sigma rejection factor
(grow   =                 0.) Rejection growing radius
(mode   =                 ql)
```

The output files will overwrite the input files. The params (trim = yes), (zerocor = yes), (darkcor = yes) are set. The param (trimsec = image) informs **ccdproc** to use the header card TRIMSEC (**lickbase** default) to perform the proper trimming. The param (readcor = yes) is set in order to reduce the noise introduced by the bias correction. This performs an unweighted "mash average" of the calibration bias frame lines, reducing the noise in the 60 Hz pick–up by the factor $\sqrt{\text{nlines}}$ (see Fig. 5–1). This single line (one dimensional) "mash average" of the calibration bias frame is subtracted line by line from the frames being bias corrected. The help file entry for "readcor" reads:

```
readcor = no (default setting)
    Convert zero level images to readout correction images?  If  yes
    then  zero  level images are averaged across the readout axis to
    form one dimensional zero level readout correction images.
```

The param (readaxi = line) is pertinent for the "zero" calibration correction provided the "readcor" param is set to "yes". The param "max_cac" is used to reduce processing time by placing the calibration frames in memory, avoiding multiple disk I/Os (useful for large data sets). Upon executing the task with ":g", the following (similar) processing messages will appear (the output order depends upon the input list order of @proclist):

```
biascal.imh: Feb  2 11:09 Trim data section is [1:2047,1:1600]
biascal.imh: Feb  2 11:09 Converted to readout format
```

```
darkcal.imh: Feb  2 11:09 Trim data section is [1:2047,1:1600]
darkcal.imh: Feb  2 11:09 Zero level correction image is biascal.imh
flatcomb.imh: Feb  2 11:09 Trim data section is [1:2047,1:1600]
flatcomb.imh: Feb  2 11:09 Zero level correction image is biascal.imh
flatcomb.imh: Feb  2 11:09 Dark count correction image is darkcal.imh
thar01.bl.imh: Feb  2 11:10 Trim data section is [1:2047,1:1600]
thar01.bl.imh: Feb  2 11:10 Zero level correction image is biascal.imh
thar01.bl.imh: Feb  2 11:10 Dark count correction image is darkcal.imh
thar02.bl.imh: Feb  2 11:10 Trim data section is [1:2047,1:1600]
thar02.bl.imh: Feb  2 11:10 Zero level correction image is biascal.imh
thar02.bl.imh: Feb  2 11:10 Dark count correction image is darkcal.imh
quartz.bl.imh Feb  2 11:10 Trim data section is [1:2047,1:1600]
quartz.bl.imh Feb  2 11:11 Zero level correction image is biascal.imh
quartz.bl.imh Feb  2 11:11 Dark Count correction image is darkcal.imh
H2Ostr.bl.imh: Feb  2 11:11 Trim data section is [1:2047,1:1600]
H2Ostr.bl.imh: Feb  2 11:11 Zero level correction image is biascal.imh
H2Ostr.bl.imh: Feb  2 11:11 Dark count correction image is darkcal.imh
data01.bl.imh: Feb  2 11:11 Trim data section is [1:2047,1:1600]
data01.bl.imh: Feb  2 11:11 Zero level correction image is biascal.imh
data01.bl.imh: Feb  2 11:11 Dark count correction image is darkcal.imh
. . .
. . .
. . .
```

8.3. Processing Status

Upon completion, check the processing status using the task **ccdlist** in the **noao imred ccdred** package, by typing "ccdlist @proclist" which gives the output:

```
biascal.imh[2047,1600][real][zero][][OT]:bias
darkcal.imh[2047,1600][real][dark][][OTZ]:dark
flatcomb.imh[2047,1600][real][flat][][OTZD]:flat
thar01.bl.imh[2047,1600][real][object][][OTZD]:Th-Ar Lamp
thar02.bl.imh[2047,1600][real][object][][OTZD]:Th-Ar Lamp
quartz.bl.imh[2047,1600][real][object][][OTZD]:flat
H2Ostr.bl.imh[2047,1600][real][object][][OTZD]:water star
data01.bl.imh[2047,1600][real][object][][OTZD]:Star X
data02.bl.imh[2047,1600][real][object][][OTZD]:Star X
data03.bl.imh[2047,1600][real][object][][OTZD]:Star X
data04.bl.imh[2047,1600][real][object][][OTZD]:Star X
data05.bl.imh[2047,1600][real][object][][OTZD]:Star X
data06.bl.imh[2047,1600][real][object][][OTZD]:Star X
```

The output format is:

```
imagename.imh[ncols,nlines][datatype][ccdtype][group][processing]:image name
```

The value of `ccdtype` is simply the value of the IMAGETYP card. Note the image dimensions `[ncols,nlines] = [2047,1600]` and the processing flags `[OTZD]`. The processing flags identifying the processing operations performed on the image are given by the following single letter codes:

```
B - Bad pixel replacement
O - Overscan bias subtraction
T - Trimming
Z - Zero level subtraction
D - Dark count subtraction
```

```
F - Flat field calibration
I - Illumination correction
Q - Fringe correction
```

The desired images have all been overscan corrected (`O`), trimmed (`T`), zero level (bias) corrected (`Z`) using the "readcor" option, and dark corrected (`D`). Their physical sizes are 2047 columns by 1600 lines, and their data types are real.

9. MODELING THE APERTURES

Before the flat field can be normalized and the program spectra can be "extracted", the locations of the echelle orders (called apertures until the wavelength calibration is complete) need to be mapped on the CCD frames. The behavior of the Hamilton is such that, at present, IRAF's *automated* routines do not satisfactorily model the apertures. This modeling is vitally important, since all further processing is based upon how well the 2D positions are modeled.

The proof of the model is the success of the spectral extraction process. If one is using a nearly full format (2000+ columns), then the curved shape of the apertures along the dispersion direction is great enough that even optimal extraction algorithms are sensitive to the aperture model. Additionally, significant variations in aperture widths and nearest neighbor separations across the echelle format make Hamilton reduction an intensive exercise in echelle data reduction.

Each aperture is simply modeled by 5 parameters: 1) an identification number, 2) an ordered pair reference coordinate (col,line) [pix], which defines its cross dispersion centroid position at a reference column of one's choosing, 3) an upper and a lower size (width) [pix] with respect to the cross dispersion profiles centroid, 4) a function whose values are ordered pair offsets (Δcol,Δline) from the reference coordinate and which describe fully the locations of each apertures cross dispersion centroids along the dispersion, and 5) an upper and a lower background region [pix] which are relative coordinates with respect to the aperture centroids.

One must be very willing to work with this step of the reduction. At anytime during later processing, one may discover that the aperture model needs "tuning". Through experience, many have learned that most problems have been remedied by fine–tuning the aperture model.

9.1. Getting to Know the apextract Package

The **noao twodspec apextract** package is a series of tasks designed to "find", "size", and "trace" echelle apertures. The meaning of these terms will be revealed in the subsequent discussion.

The help for **apextract** reads:

```
                    noao.twodspec.apextract
        apall - Extract 1D spectra (all parameters in one task)
    apdefault - Set the default aperture parameters and apidtable
      apdemos - Various tutorial demonstrations
       apedit - Edit apertures interactively
       apfind - Automatically find spectra and define apertures
        apfit - Fit 2D spectra and output the fit, difference, or ratio
    apflatten - Remove overall spectral and profile shapes from flat fields
       apmask - Create and IRAF pixel list mask of the apertures
  apnormalize - Normalize 2D apertures by 1D functions
   aprecenter - Recenter apertures
     apresize - Resize apertures
    apscatter - Fit and subtract scattered light
        apsum - Extract 1D spectra
      aptrace - Trace positions of spectra
```

with the additional help topic files:

```
apbackground - Background subtraction algorithms
  approfiles - Profile determination algorithms
  apvariance - Extractions, variance weighting, cleaning, and noise model
     package - Package parameters and general description of package
```

If one really desires to be familiar with the operations involved with aperture modeling, one should peruse the help files. For the process of "finding" apertures, it is helpful to read the **center1d** help file, which describes an **xtool** used for locating the centroid of a peaked distribution (like the cross dispersion illumination profile of an echelle aperture!). Additionally, F. Valdes of NOAO has written *The IRAF APEXTRACT Package* and several revision summaries. These guides through the **apextract** package are available via the Internet (URL=ftp://iraf.noao.edu/iraf/docs). There is a plethora of other information as well.

The task **apall** is a generalized task that lets one avoid a maze of "hidden" parameter settings. It is a driver that invokes the above listed tasks with a single parameter set.

9.2. The Aperture Game Plan

Here, the overall plan to obtain "good" models of the apertures is outlined:

(1) Find the apertures. With the **center1d** algorithm, the **apall** task will locate the cross dispersion centroids of the apertures along a single "reference" column. This process is sensitive to several input parameters. One can interactively modify (add, delete, recenter, renumber...) the automated result in **apall** editing mode. However, if one's first attempts are way off, one should bail out and intelligently reset the finding input parameters.

(2) Size the apertures. During the finding process, the aperture cross dispersion widths can either be automatically determined or be "hard–wired" by setting certain input parameters. Again, one can interactively modify the automated result.

(3) Set the aperture background regions. This step requires interactive editing (for the Hamilton). The troughs between apertures contain both background scattered light and a non–negligible percentage of overlap from nearest neighbor apertures. The extraction of the spectra is quite sensitive to the definition of these regions. Also, one sets background fitting parameters, such as sigma rejections, fitting function, etc.

(4) Trace the apertures. Once one is satisfied that the aperture centroids are well determined along the reference column, one needs to map them in the dispersion direction. One aperture at a time, one will be placed in **icfit** mode with plots of the pixel verses pixel positions of the aperture centroids. One will fit a polynomial or spline curve to these as a model describing the full location of the aperture on the data frame.

Once an aperture "model" is constructed, **apall** will create a "database" subdirectory in which a file called "apimagename" will store the aperture models. This model may be applied to other images in the data set or even other data sets and then adjusted using **apall**.

9.2.1. On Filters, Light Paths, and Apertures

For a particular instrumental setting, apertures locations and sizes are a function of the light path through the instrument. If one uses different filters for an exposer, the different light paths will result in slightly shifted apertures. Moreover, the light path for the Quartz Lamp is not identical to that of objects seen through the telescope. In other words, the Quartz apertures will not identically map those of the program data, as illustrated in Fig. 9–1.

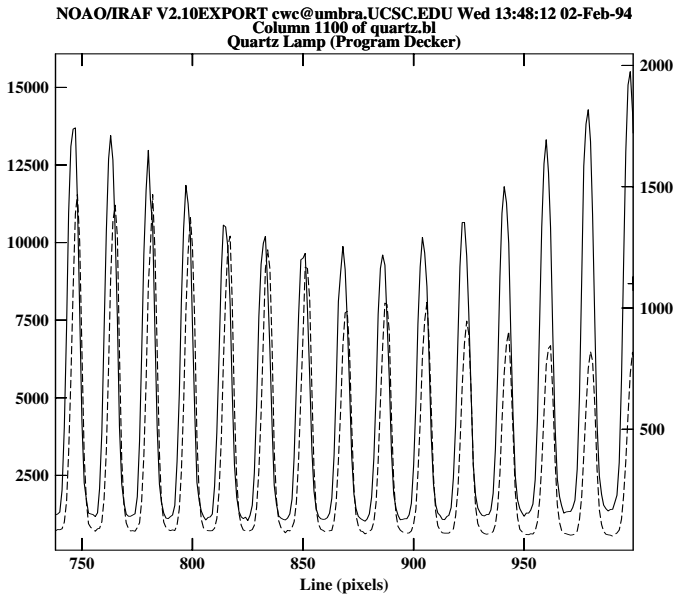


Figure 9–1. Several cross dispersion profiles are plotted, showing their locations along column 1100. The Decker setting is the same as used for the program data. Note that the aperture positions of the Quartz frame (solid lines), which was exposed through bg12 and bg13 filters, are shifted from the aperture positions of the water star (dashed lines), which was exposed using no filters.

9.2.2. Using an Aperture Template Frame

To ensure that the process of locating aperture cross dispersion centroids is robust, one should use a high–signal low–feature frame. Often, the program data do not satisfy these criteria and one may choose to use an "aperture template" frame, one which does satisfy the criteria. However, any aperture template frame should have been exposed with as similar a light path as possible to that of the program data.

The frame that best meets the former criterion is the Quartz frame. However, it is difficult to obtain uniformly high–signal Quartz frames without filters (due to the CCD QE blue roll–off) and the Quartz Lamp light path is not so similar to that of the program data. Thus, the Quartz frame does not meet the latter criterion very well. However, the water star does, and depending upon its spectral features, it may satisfactorily meet the former criterion as well.

Ultimately, the choice for the aperture template is a trade–off of these concerns and the amount of fine tuning of the aperture model one wishes to perform. Aperture models based upon the Quartz frame will require shifting (recentering) when applied to the program data. However, the centroid locations and sizes (widths) will be very robust making the modeling process quick and easy, though

the sizes may be slightly large and may also require adjustment. Aperture models based upon the water star may be more difficult and tenuous, since the centroid locations may not be as cleanly determined. Moreover, one must be careful defining the sizes since absorption or emission features located on the reference column will misrepresent the cross dispersion profile sizes.

For the example here, the Quartz frame will be used as the aperture model template. Then, a few fine tuning steps will be performed while applying the model to both the calibration flat field frame (recall, this frame has wide apertures) and the program data. Consider the following steps:

(1) Automatically find and size the apertures of the Quartz frame. Interactively trace the centroids in the dispersion direction. Save the model to the database.

(2) Modify the Quartz model for the calibration flat field frame: interactively size the apertures; interactively set the background parameters. Save this model to the database.

(3) Modify the Quartz model for the water star, program data, and arcs. Using the water star, automatically recenter the aperture reference coordinates, interactively tuning some of these new locations if needed. Interactively edit the sizes as needed. Interactively set the background parameters. Save this model to the database.

The step in which one applies these models to the individual frames is performed during the extraction process.

9.2.3. Preparing for apall

Before beginning, use **display** to visually inspect the Quartz frame apertures to estimate the number of apertures, an average aperture separation in pixels, and an average aperture width in pixels. At this point, these values are "defaults", so one should not bother attempting to measure them better than ± 1 pixel (do not invest a great deal of time determining them). In general, the apertures are closer together in the red and further separated toward the blue.

The important values to obtain before modeling the template apertures are (1) a good choice of a reference column for the reference coordinate of each aperture (no blemishes or spectral features and few cosmic rays), (2) a decent measure of the cross dispersion centroid separation (on the reference column) of the two closest apertures, and (3) the peak value of the lowest signal aperture (on the reference column). When choosing the reference column, select one near the center of the frame that has no cosmic rays with intensity greater than the lowest signal aperture, if possible.

If one is using the Quartz frame as a template, there is one more useful preliminary measurement. Attempt to determine the relative sizes (widths) of the Quartz and water star apertures. In particular, one may find that the widths of the water star apertures are smaller than those of the Quartz. Quantitatively, the water star widths may be (for example) equal to the 20% peak level widths of the Quartz. In other words, after one "visually" sets the zero point at the interorder values and measure the Quartz widths 20% up to the aperture peaks, one will find that these widths are equivalent to the baseline widths of the water star. When one applies the Quartz frame aperture models to the water star, one may find that this knowledge may save time and interactive editing steps.

9.3. Getting the Template Aperture Model

Having obtained the preliminary quantities outlined above, epar into **apall** and set the param-

eters: (below is only a partial listing of the pertinent parameters):

```
PACKAGE = echelle
   TASK = apall
input   =             quartz.bl  List of input images
(output =                     ) List of output spectra
(format =               echelle) 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=                    no) Recenter apertures?
(resize =                   yes) Resize apertures?
(edit   =                   yes) Edit apertures?
(trace  =                   yes) Trace apertures?
(fittrac=                   yes) Fit the traced points interactively?
(extract=                    no) Extract spectra?
(extras =                    no) Extract sky, sigma, etc.?
(review =                    no) Review extractions?
(line   =                  1100) Dispersion line
(nsum   =                     5) Number of dispersion lines to sum
                                 # DEFAULT APERTURE PARAMETERS
(dispaxi=                     1) Dispersion axis 1=along lines, 2=along columns
(lower  =                    -5.) Lower aperture limit relative to center
(upper  =                     5.) Upper aperture limit relative to center
(apidtab=                     ) Aperture ID table (optional)
                                 # DEFAULT BACKGROUND PARAMETERS
(b_funct=              legendre) Background function
(b_order=                     2) Background function order
(b_sampl=                     ) Background sample regions
(b_naver=                    -1) Background average or median
(b_niter=                     3) Background rejection iterations
(b_low_r=                    3.) Background lower rejection sigma
(b_high_=                    3.) Background upper rejection sigma
(b_grow =                    0.) Background rejection growing radius
                                 # APERTURE CENTERING PARAMETERS
(width  =                    5.) Profile centering width
(radius =                    5.) Profile centering radius
(thresho=                  250.) Detection threshold for profile centering
                                 # AUTOMATIC FINDING AND ORDERING PARAMETERS
nfind   =                    92  Number of apertures to be found automatically
(minsep =                   10.) Minimum separation between spectra
(maxsep =                  100.) Maximum separation between spectra
(order  =            increasing) Order of apertures
                                 # RESIZING PARAMETERS
(llimit =                INDEF) Lower aperture limit relative to center
(ulimit =                INDEF) Upper aperture limit relative to center
(ylevel =                  0.2) Fraction of peak intensity for automatic wid
(peak   =                  yes) Is ylevel a fraction of the peak?
(bkg    =                  yes) Subtract background in automatic width?
(r_grow =                   0.) Grow limits by this factor
(avglimi=                   no) Average limits over all apertures?
                                 # TRACING PARAMETERS
(t_nsum =                    4) Number of dispersion lines to sum
(t_step =                   15) Tracing step
(t_nlost=                    4) Number of consecutiv times profile is lost
(t_funct=              spline3) Trace fitting function
(t_order=                    3) Trace fitting function order
(t_sampl=                    *) Trace sample regions
(t_naver=                    1) Trace average or median
(t_niter=                    4) Trace rejection iterations
(t_low_r=                  2.5) Trace lower rejection sigma
(t_high_=                  2.5) Trace upper rejection sigma
(t_grow =                   0.) Trace rejection growing radius
```

The task is run interactively with the params (interac = yes), (find = yes), (edit = yes), (resize = yes), (trace = yes), and (fittrac = yes) set. The "line" parameter is set to column 1100. This is the reference column on which the "search" for the aperture will occur and their centroids will be centered. Recall, that one will need to pay attention to cosmic rays present on this column, for they can fool **apall** into thinking it has found an aperture (see below). The choice for "line" accounts for good signal strength on all apertures and no CCD blemished along that column. If one leaves "line" as the default 'INDEF', then **apall** uses the center column of the image. The "nsum" parameter gives the number of columns **apall** should add together centered on "line" while finding the aperture peaks. The larger this number the better the peaks are defined statistically, but "nsum" should be no larger than around 5 so that the curvature of the apertures does not skew the peaks off their true centers.

The *default aperture* parameters (lower = $-5.$) and (upper = 5.) set the default widths of the apertures in pixels with respect to the aperture center as determined by **apfind** and **center1d**. This choice was made for a 2.5" Decker. These parameters are actually inconsequential, since we are going to let **apall** determine the sizes of the apertures for us (eg. "resize = yes" '). However, the defaults parameters should be set to some overall characteristic aperture width. Though the parameter (dispaxi = 1) is already defined in the image header card DISPAXI, provided one instructed **lickbase** to insert it, set it to '1'.

The *default background* fitting parameters should be set now. However, leave the "b_sampl" field blank, since we are going to interactively insert these aperture by aperture in the following steps. *The background settings are used only during the profile fitting and spectra extraction processes.* All other processes claiming to perform "background" subtraction do not key off "b_sampl", they simply use the first local minima in the cross dispersion direction of the current aperture! Set the function to a polynomial with (b_order = 2). Set the rejection iterations to at least (b_niter = 3).

The *aperture centering* parameters actually are inputs for the **xtool** package **center1d** task. This task convolves a saw–tooth function with the aperture cross dispersion profiles (bell–shaped distribution) to determine their centroids. The help file for **center1d** has a clear graphical illustration of the meaning of the following parameters and explains the centering process very clearly. Together, the params (in pixels) (width = 5.) and (radius = 5.) define the limits of integration for the convolution integral and the limits for determining the continuum "intensity" (background). It is effective to choose "width = average width of apertures" and "radius = width". This forces the centering routine to stay on the aperture while giving it the entire aperture to perform its centering. Too narrow a width results in decreased accuracy. For the 2.5" Decker setting, "width = radius = 5" pixels gives $A = 7.5$ (the $\pm$ limits for determining the surrounding "continuum") and $B = 12.5$ (the limits of the centering convolution integral). This combination seems to result in well centered apertures. Very important is the "thresho" parameter. This should be set to a value significantly less than the peak of the dimmest aperture and (if possible) significantly greater than the height of any cosmic rays present on the reference column, "line".

The *automatic finding and ordering* parameters include "nfind", which is a query parameter (all parameters in the epar files that do not have a closed ")" are run time query parameters), and should be set equal to or slightly greater than the number of real apertures. It is best to have counted apertures using an image **display**. The param (minsep = 10.) is set to be about 2/3 the minimum separation of the two closest apertures. This parameter is quite important. If it is set too small, **apall** tends to locate "apertures" in the interorder regions. If it is too large, **apall** will simply skip real apertures. The param (maxsep = 100.) may be set to any number greater than the aperture peak to peak separations. The results are quite insensitive to this parameter. For the Hamilton, set (order = increasing). During wavelength calibration, the task **ecidentify** will attempt to determine the offset between model aperture number and true order number.

The *resizing parameters* are set such that the width of each aperture will be defined by its width at 20% of the peak value (per the above water star discussion). For this purpose, the params (llimit

= INDEF) and (ulimit = INDEF) must be set. The param (ylevel = 0.2) and (peak = yes) dictate the percent of peak intensity at which the width is automatically sized. Critical to success is the param (bkg = yes), which defines the adjacent interorder minima as the zero point for measuring the peak values. The automatic sizing fails without this. (Actually, the routine uses the first local minima in the cross dispersion direction of the current aperture. For this reason, automatic sizing works for the Quartz frame, but not for the wide Decker flat frame. The flat field has structure across the aperture, so this "background" determination fails, resulting in meaningless aperture widths).

The *tracing parameters* "t_nsum", "t_step", and "t_nlost" are quite important, since these parameters *cannot be adjusted* during interactive tracing (argh!). Set (t_nsum = 3), which makes tracing more robust near the CCD edges and for weak apertures. Setting "t_nsum" to '1' may result in confusion that cannot be remedied without aborting! Setting "t_nsum" to a much larger number is not recommended since this results in a "blurred" position. The tracing step (t_step = 15) is a critical parameter. Usually, wild difficulties can be remedied by lowering this parameter (but do not rule out enlarging it!). So that the trace doesn't jump orders or behave too crazy, set (t_nlost = 4), never higher. This tells the braille method tracing routine to bail if it is confused as to the position of the present aperture 4 tracing steps in a row. In the interactive tracing mode, one will have the flexibility of modifying the fitting function, the function order, the sample region, the number of points to average, the rejection iterations, the rejection sigma, and the rejection growing radius (see the **aptrace** help file for definitions). The function param (t_funct = spline3) is a most flexible fitting model with (t_order = 3). One may wish to tighten the sigma to ± 2.5. It is always good to have several rejection iterations, otherwise bad points will not be excluded from the fit.

(A note about responding to the interactive prompts: These prompts may be answered individually with the lower case responses "yes" or "no" or answered for all further prompts with the responses "YES" or "NO". Note that answering "YES" or "NO" to an aperture prompt applies to all further apertures, whereas such response to a prompt concerning the frame as a whole applies to all further frames. When an aperture is not fit interactively the last set of fitting parameters are used, not the defaults set in the epar file.)

Upon executing **apall** the following prompts appear with the defaults in ():

```
Find apertures for quartz.bl?  (yes):
Number of apertures to be found automatically (92):
Resize apertures for quartz.bl?  (yes):
Edit apertures for quartz.bl?  (yes):
```

Just hit [CR] for each of the above. After several seconds the plotting window will display the cut along the reference column "line" with a statement such as

```
aperture = 1  beam = 1  center = 16.51   low = -3.92  upper = 3.78
```

One is now in edit mode. The current aperture is 1, which has reference coordinate (1100,16.51) with automatically determined lower and upper widths of -3.92 and 3.78 pixels, respectively. As shown in Fig. 9–2, the graph will be very busy and compressed. Across the top are aperture markers labeled with the aperture number. Use the "window" commands to navigate about the graph and visually inspect the aperture assignments. Since the keystrokes are now set for **apedit**, one must actually type "w" to enter into window mode. The most useful window commands are the "we"–"e" (expand) keys. Fig. 9–3 shows a windowed region of Fig. 9–2. Typing "wa" will always return one to the original plot. If IRAF just beeps or the plot starts doing funny things, type [CTRL–C] and

then a slow series of "?"s until a beep is heard. Type "wa" or "?". All should be back to normal.

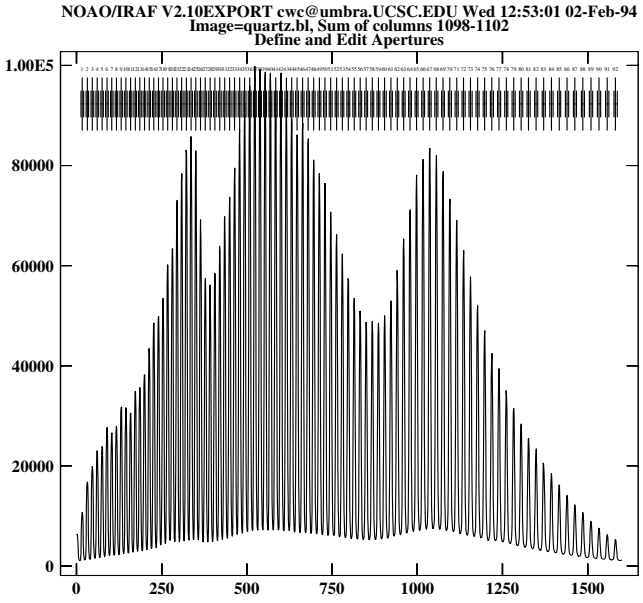


Figure 9–2. A full window plot of the aperture locations centered along column 1100 is very busy. The hash marks across the top are where **apall** has centered the apertures along the reference column. The "error" bars give the upper and lower widths that **apall** determined using the "ylevel" resizing parameter.

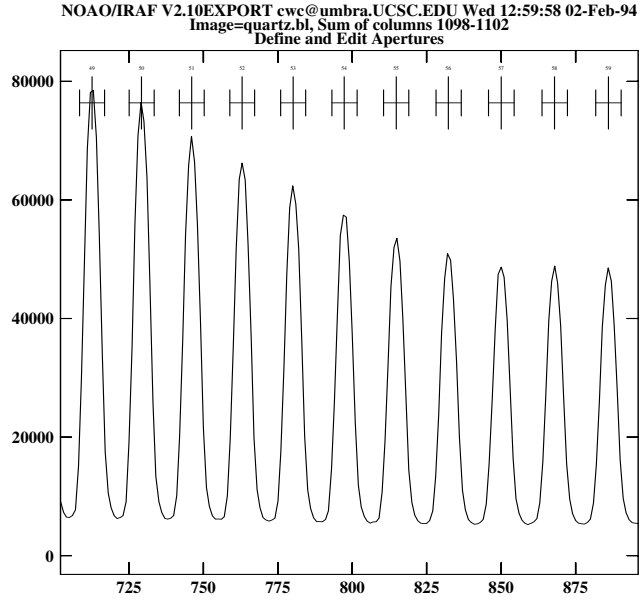


Figure 9–3. By using the "window" commands, one can zoom the plot to visually examine the aperture definitions. The model shown here is highly satisfactory.

The aperture assignments should be satisfactory. With the Quartz frame, the model should be right the first time if the input parameters are set correctly. If they are hopelessly wrong, one can abort

(type "I") and reset the **apall** parameters based upon these results.

> Whenever one aborts a task, always do the following, or one may experience strange results:
>
> (1) Flush the tasks process using **flprc**
>
> (2) **imdelete** any output file the task created

If however, the model looks good, but with a few blemishes, one can modify them to one's choosing in the **apall** editor. To get a list of helpful editing commands, type "?". As with all interactive modes, practice is the key. However, editing the apertures is particularly demanding until one gets the knack. Some most useful commands include:

```
?    Print help
a    Toggle the ALL flag
b an Set background fitting parameters
d an Delete aperture(s)
g an Recenter aperture(s) (see APRECENTER)
l ac Set lower limit of current aperture at cursor position
m    Define and center a new aperture on the profile near the cursor
n    Define a new aperture centered at the cursor
o  n Enter desired aperture number for cursor selected aperture and
     remaining apertures are reordered using apidtable and maxsep
     parameters (see APFIND for ordering algorithm)
q    Quit
r    Redraw the graph
u ac Set upper limit of current aperture at cursor position
w    Window the graph using the window cursor keys
y an Set aperture limits to intercept the data at the cursor y
     position
z an Resize aperture(s) (see APRESIZE)
.    Select the aperture nearest the cursor for current aperture
+  c Select the next aperture (in ID) to be the current aperture
-  c Select the previous aperture (in ID) to be the current aperture
I    Interrupt task immediately.  Database information is not saved.
```

Some of these prompt for an aperture number, some instantly act upon the "current aperture", and some act upon the aperture nearest the cursor. It is always safest to define the aperture of interest as the current aperture with ".", and then perform an editing operation. After any operation, type "r", to update the plot. To edit all apertures simultaneously, type "a". This key *toggles* "ALL" mode; any following operation on any single aperture is applied to all apertures. When finished , type "q", following which the prompt will appear

```
Trace apertures for quartz.bl?  (yes):
Fit traced positions for quartz.bl interactively?  (yes):
```

Upon [CR]s, **apall** will trace the apertures along the dispersion direction outputting messages like:

```
Oct 11  9:49: TRACE - Trace of aperture 1 in quartz.bl lost at col 1940
Oct 11  9:49: TRACE - Trace of aperture 1 in quartz.bl lost at col 1975
Oct 11  9:49: TRACE - Trace of aperture 1 in quartz.bl recovered at col 2010
Oct 11  9:49: TRACE - Trace of aperture 1 in quartz.bl lost at col 2045
Fit curve to aperture 1 of quartz.bl interactively  (yes):
```

Upon [CR], a line verses column plot of the aperture centroids will be plotted and one will be in interactive mode. One is now in control over the function fit via **icfit**.

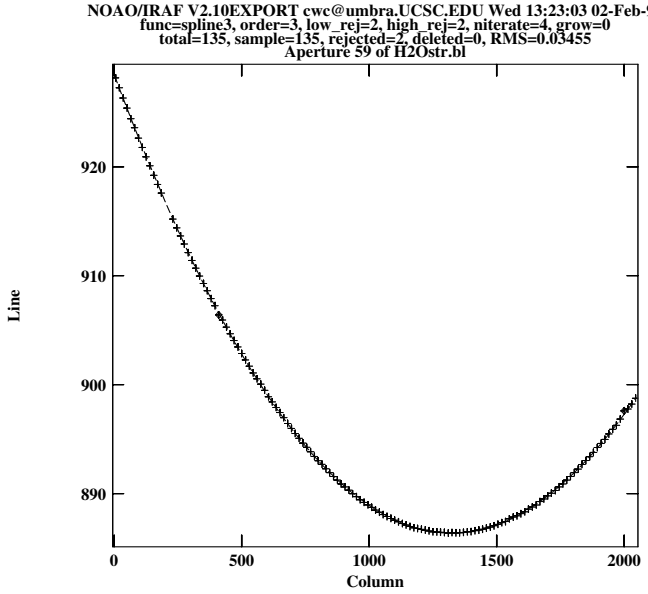


Figure 9–4. The aperture tracing step is performed interactively. One by one, set the fitting parameters to line verses column plots of the apertures. Shown here is how the fit has jumped over a bad region of the CCD along aperture 59. A total of 2 points have been rejected from this 4th iteration on the fit.

During the tracing process, never change the plotting window via the "graph keys". This seems to confuse the tracing algorithm, and one may be unable to recover. To apply the fit and move to the next aperture, type "q". If one has many apertures, one may wish to allow **apall** to finish the job automatically. Thus, after doing a few to several apertures, at the next prompt one may type "NO", which will apply the last settings to the remaining apertures. However, it is not recommended one does this if the last aperture "falls" off the frame. In this case, one obtains a "singular solution". This is no problem. The way to handle these partial apertures is to lower the fitting function to a 2nd order polynomial, eliminating wild un–constraint. When **apall** returns one will be prompted:

```
Write apertures for quartz.bl to database  (yes):
```

Upon [CR], the locations, sizes, and tracings of the apertures will be saved in the file "apquartz.bl" in the "database" subdirectory. The entry for each aperture will look something like:

```
# Fri 21:49:29 10-Sep-93
begin   aperture quartz.bl 1 1100. 16.50628
        image   quartz.bl
        aperture        1
        beam    1
        center  1100. 16.50628
        low     -1099. -3.92
        high    947. 3.78
        background
                xmin -5.
                xmax 5.
                function legendre
                order 2
```

```
                sample
                naverage -1
                niterate 3
                low_reject 3.
                high_reject 3.
                grow 0.
        axis    2
        . . .
        . . .
        . . .
```

followed by the coefficients of the aperture tracing function.

9.4. Tailoring the Aperture Model for the Flat Field

Except for the Decker setting, the calibration flat field frame was exposed under the same conditions as the Quartz frame. We will apply the Quartz aperture model to the flat field, but we will interactively define the sizes and background sample regions. The issues at hand are setting the aperture widths accurately, and setting the background sample regions accurately. Again, epar into **apall** and set the parameters for interactive editing (partial listing – no *finding, centering, sizing* or *tracing* parameters will be invoked since they are all taken from the Quartz model):

```
PACKAGE = echelle
   TASK = apall
input   =              flatcomb  List of input images
(output =                      ) List of output spectra
(format =               echelle) Extracted spectra format
(referen=             quartz.bl) List of aperture reference images
(profile=                      ) List of aperture profile images
(interac=                   yes) Run task interactively?
(find   =                    no) Find apertures?
(recente=                    no) Recenter apertures?
(resize =                    no) Resize apertures?
(edit   =                   yes) Edit apertures?
(trace  =                    no) Trace apertures?
(fittrac=                    no) Fit the traced points interactively?
(extract=                    no) Extract spectra?
(extras =                    no) Extract sky, sigma, etc.?
(review =                    no) Review extractions?
```

The input frame is the calibration flat field frame and the aperture model is (referen = quartz.bl). Note that only (interac = yes) and (edit = yes). The remainder of the package parameters will not be used. Upon ":g" one will be prompt to edit the aperture parameters. Hit [CR] to obtain a plot similar to Fig. 9–2. Window in closely to see that the aperture widths are too narrow, but that the centers correspond quite well (see Fig. 9–5). Remember, in interactive mode always type keystrokes slowly and deliberately.

Note the jaggies near the aperture peaks. It is due to these "features" that resizing the flat field apertures must be performed interactively. If the resizing routine was more intelligent, meaning it didn't assume that the first local minimum in the cross dispersion direction was the aperture edge, then this entire interactive process could be automated. The resizing routine allows one to automatically set an average width over all apertures, but this has normally results in entirely unsatisfactory normalization of the flat field.

9.4.1. Interactively Resizing Aperture Widths

Stepping through each aperture, one will use the "y" option of the editing keystrokes to set the widths. The most useful **apedit** commands for this include:

```
?    Print help
q    Quit
r    Redraw the graph
u    Set upper limit(s)
l    Set lower limit(s)
w    Window the graph using the window cursor keys
I    Interrupt task immediately.  Database information is not saved.
y    Y level limit(s)
```

though one may almost exclusively use the "w" and "y" commands.

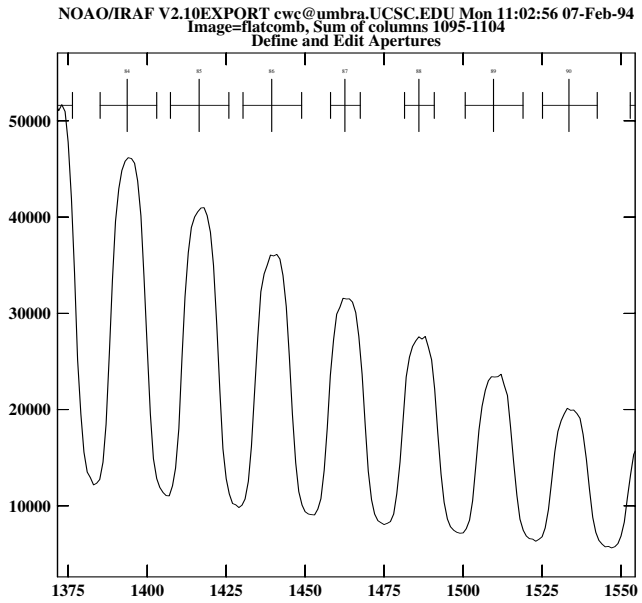


Figure 9–5. The windowed region of "flatcomb" centered along column 1100 shows the aperture widths. Here, apertures 84, 85, 86, 89, and 90 have been manually resized using the "y" key in edit mode. Apertures 87 and 88 have not yet been edited, and are still the size input by the reference model "quartz.bl".

As shown in Fig. 9–5, window into a region of 5 or 6 aperture cross sections. Place the cursor within the *defined* area of a given aperture at the base of the profile, where the intensity level roughly represents the aperture cross dispersion extremes. Type a "y". The "error bar" defining the aperture size will expand to a more appropriate width. Type an "r" to redraw a clean graph. As one proceeds, the **apedit** output will look something like:

```
aperture = 84  beam = 84  center = 1393.65  low = -8.57  upper = 9.33
aperture = 84  beam = 84  center = 1393.65  low = -8.57  upper = 9.33
aperture = 85  beam = 85  center = 1416.44  low = -9.04  upper = 9.43
aperture = 86  beam = 86  center = 1439.42  low = -9.11  upper = 9.46
aperture = 89  beam = 89  center = 1509.63  low = -8.99  upper = 9.36
aperture = 90  beam = 90  center = 1533.48  low = -8.38  upper = 8.97
```

Repeat this step for all apertures. Before moving on to the next step of setting the background, be

sure the results are satisfactory.

9.4.2. Interactively Setting Aperture Background

This step is very intensive. One will always need to be well focused on particular regions during this process. As before, center the cursor within the defined area of a given aperture, this time type "b". Now in **icfit** mode, one will see an expanded plot in relative coordinates with the origin at the aperture center as illustrated in Fig. 9–6. The complaint "Warning: No sample points for fit" will appear (if you left the "b_sampl" parameter blank), which can be ignored. Window this plot to expand around the current aperture. (If one needs to re–window and type "wa", don't be alarmed when the plot is expanded out to the entire aperture format. Simply window back in, step by step.)

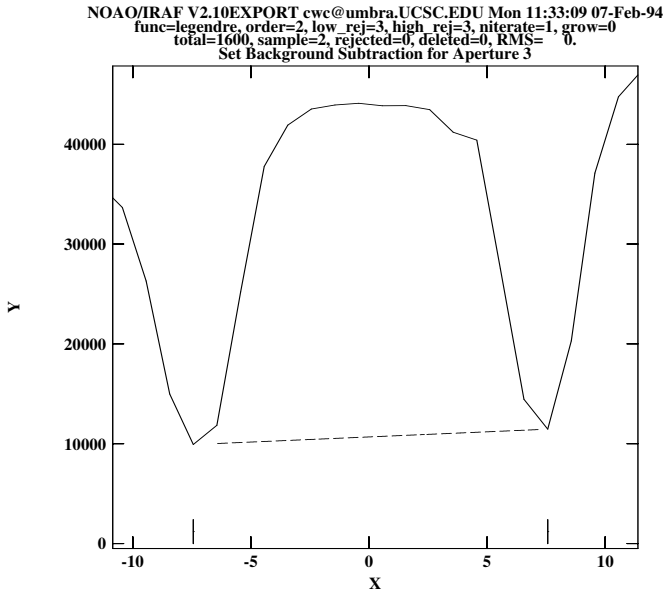


Figure 9–6. Following the keystroke "b" within aperture 3, the following plot will appear and one will be in the **icfit** routine. Use the "s" keystroke to interactively define the "sample" parameter with the cursor. One may perform all the functions of **icfit**, including examination of the fit, as shown here.

The background sample should basically be the inter–order minima, as many pixels wide as one deems appropriate. The lower numbered apertures are basically over–lapped, so the sample range may be only a fraction of a pixel! (This works.) To set the background sample region, use the "s" key, setting the lower limit of the region at the cursor position. One will be prompted with "again:". Move the cursor to the upper limit and type "s" again. An sideways "error bar" should appear where the sample has been defined (if the sample is a fraction of a pixel it will be a single vertical bar, as shown in Fig. 9–6). Now repeat, putting a sample region on the other side of the aperture.

At anytime, to undefine the sample closest to the cursor, type "z". To totally clear the sample regions, type ":sample ∗". Typing ":sample" will output something like

```
sample =  -7.948311:-7.287847 7.371423:7.857282
```

The vertical bar may not always appear. In such cases, upon typing "f", one will note this

defined sample will not be included in the fit, even though the sample is clearly defined as revealed by typing ":show" or ":sample". This happens when the selected sample region lies within the upper and lower limits of the current aperture. For the former one may need to "q" out of **icfit** and back into editing mode to redefine the width with the "y" key, making it a bit narrower, and then type "b" to return to background fitting. This iterative process should result in very well defined aperture widths and background sample regions.

Upon quitting, be sure to save the aperture model to database. If one wishes to get fancy with the background, one may set several samples regions, including more than just the nearest neighbor interorder minima. If so, one may want to change the default fitting parameters to be more flexible than the simple 2nd order polynomial that is used in this example.

9.5. Tailoring the Aperture Model for the Program Data

Having the same Decker setting and a similar optical path through the telescope and instrument, the low-feature water star is a good frame for adjusting the model apertures for the program data. If one accounted for the relative smaller sizes of the water star apertures when automatically sizing the Quartz apertures, than one does not need to resize. In this step, one will interactively recenter the aperture centroids along the reference column using the water star and set the background sample regions. There is no evidence that one needs to retrace the apertures. The overall curvature in the dispersion direction is seemingly unaffected by slight changes in the light path. Epar into **apall** and set the parameters (partial listing):

```
PACKAGE = echelle
   TASK = apall
input   =             H2Ostr.bl  List of input images
(output =                      ) List of output spectra
(format =               echelle) Extracted spectra format
(referen=             quartz.bl) List of aperture reference images
(profile=                      ) List of aperture profile images
(interac=                   yes) Run task interactively?
(find   =                    no) Find apertures?
(recente=                   yes) Recenter apertures?
(resize =                    no) Resize apertures?
(edit   =                   yes) Edit apertures?
(trace  =                    no) Trace apertures?
(fittrac=                    no) Fit the traced points interactively?
(extract=                    no) Extract spectra?
(extras =                    no) Extract sky, sigma, etc.?
(review =                    no) Review extractions?
```

Following ":g" and confirmation of the recentering and editing prompt, window around the edit plot. One may notice that some aperture widths are wider than the base of their cross dispersion profile. For the most part this occurs where an absorption feature has been intercepted (is the peak level lower than the nearest neighbor peak levels?). Be sure to scrutinize thoughts about resizing these apertures. Additionally, be sure to account for this absorption feature effect when setting the background samples; do not get too greedy and push the sample up into the aperture edge.

Follow the steps outlined above for setting the background sample for the flat field. One should be relieved at the relative amount of inter–order real–estate in the blue (large aperture numbers). As mentioned, do not get greedy. Choose only a few pixel wide swath across the minima, as shown

in Fig. 9–7. Upon completion, type "q" and save the aperture model to database.

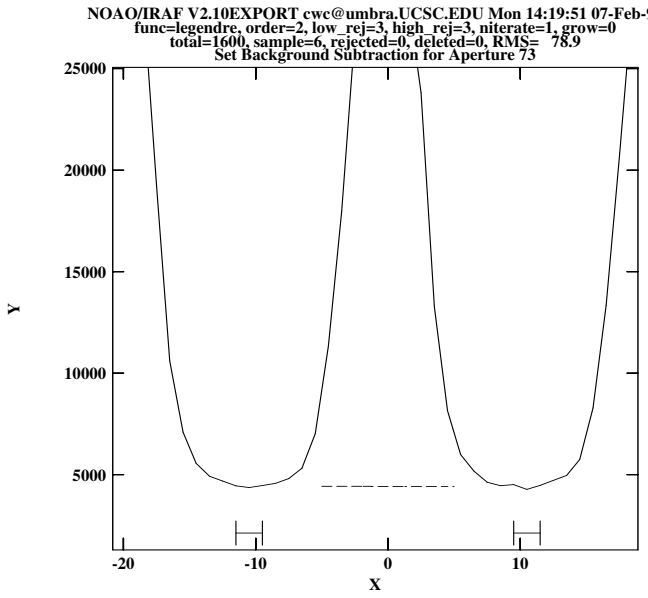


Figure 9–7. There is more real–estate between the apertures for the program Decker setting. Do not over expand these areas, but stay close to the minima. If the present aperture or the nearest neighbor is an absorption feature cross section, take this into account also.

9.6. What is the Background?

Depending upon one's spectrograph, the meaning and the use of the background can be adjusted. For spectrographs with wide order separations, a proper Decker setting will sample the sky at the aperture cross dispersion extremes. In this case, the "background" setting would be judiciously set to remove the sky spectrum from the program spectrum during the extraction process. Such is not the case for the Hamilton. One cannot properly remove sky contamination from Hamilton data using this technique.

With the Hamilton, as with other echelle spectrographs with tightly packed orders, the "background" could be used to subtract "scattered light" or simply define the intensity zero point (the **apscatter** task is specifically designed for this approach). Effectively, this is the approach for the flat field frame as presented here (see §10.2 for more on this). However, with the Hamilton, the apertures overlap, so the zero point is not accurately modeled using this technique.

If one is measuring absolute quantities, such as equivalent widths, one has a serious problem, since the results will be sensitive to the zero point determination. This will be briefly addressed in §12.1 and §13 and is discussed in Appendix C. Also, see Churchill and Allen (1995) for details on determining the Hamilton zero point.

10. NORMALIZING THE FLAT FIELD FRAME

Now that the aperture database is constructed, the calibration flat field frame can be normalized. The object here is to create a frame used to divide out the pixel to pixel gain variations. Normalizing the flat field requires modeling the aperture illumination profiles and then dividing them by smoothed versions of the profile models. This modeling is not trivial, and is quite sensitive to the aperture model. Indeed, one may find that the aperture model may need slight adjustments before a satisfactory normalization is achieved.

10.1. Choosing the Flattener Task

IRAF provides two general techniques for normalizing the illumination profiles from the apertures. The first is the task **apnormalize**, which simply removes the blaze function, the illumination profile in the dispersion direction. The second is the task **apflatten**, which removes both the blaze function and the cross dispersion illumination profile. These two tasks are well described in the respective help files and in further detail in *The IRAF APEXTRACT Package* (Valdes 1990). In addition, the **apflatten** topic file describes three flat fielding "philosophies" for echelle data.

The **apnormalize** task simply normalizes the 2D illumination profiles by 1D functions along the dispersion direction. The primary purpose of this approach is to remove the blaze function while retaining the cross dispersion profile shape. The 2D profiles are extracted into 1D spectra (interactively if desired), which are then smoothed and scaled into 1D functions. These functions are either scaled to the peak intensity of the cross dispersion profile to obtain unity at the aperture centers (this has a V2.10.2 bug, see Appendix A), or scaled to the mean intensity of the cross dispersion profile to obtain unity closer to the aperture edges. The data are divided by these 1D "blaze" functions; all data outside the defined apertures are set to unity.

In the **apflatten** task, the illumination profiles are modeled in both the dispersion and cross dispersion directions. The 2D profiles are "optimally" extracted into 1D spectra and smoothed along the dispersion direction (interactively if desired). The optimal extraction routine models the cross dispersion profiles as normalized probability functions (Horne 1986; Marsh 1989). These 2D models are then divided into their respective apertures, replacing points of "low" signal within the aperture and all points outside the aperture by unity. The result is a pixel to pixel "sensitivity variation" flat field image.

The cross dispersion profile optimal fitting algorithm is described in **approfiles**, **apvariance** and **apbackground** topic files of **apextract**. Also, see Appendix B, which outlines variance weighted optimal extraction.

10.2. Modeling the Illumination Profiles

The **apflatten** task is highly recommended as the flat field normalizing algorithm. However,

be forewarned; obtaining well normalized apertures is very difficult. Several artifacts from the normalization process may plague the flat field data. The most common unwanted features following normalization are edge effects that ripple along the dispersion direction. These are difficult to remove completely. However, one can successfully minimize them with good aperture models. Occasionally, one to a few normalized apertures will be unacceptable, exhibiting rope–like twisting down the dispersion direction. In this case, repeat **apflatten** with (edit = yes). In edit mode, modify the aperture models for these "bad" apertures before launching back into fitting mode. How does one know what modifications to make? Usually the bad apertures are due to under estimating the aperture widths or over estimating the background regions. Examine the models of the apertures which successfully normalized, using them as a guide. One may need to iterate this process of adjusting and flattening.

Within **apflatten**, there are many options from which to choose for constructing the illumination profile models, all of which are explained in detail in the help files. The most satisfactory and consistent results to date have been obtained using the aperture model as outlined in §9 with a 2D optimal profile fitting routine. Epar into **apflatten** and set the parameters:

```
PACKAGE = echelle
   TASK = apflatten
input   =              flatcomb  List of images to flatten
output  =             flatfield  List of output flatten images
(referen=                     ) List of reference images
(interac=                   yes) 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 traced points interactively?
(flatten=                   yes) Flatten spectra?
(fitspec=                   yes) Fit normalization spectra interactively?
(line   =                 INDEF) Dispersion line
(nsum   =                     1) Number of dispersion lines to sum
(thresho=                   10.) Threshold for flattening spectra
(backgro=               average) Background to subtract
(pfit   =                 fit2d) Profile fitting type (fit1d|fit2d)
(clean  =                    no) Detect and replace bad pixels?
(skybox =                     6) Box car smoothing length for sky
(saturat=                 INDEF) Saturation level
(readnoi=               rdnoise) Read out noise sigma (photons)
(gain   =                   1.0) Photon gain (photons/data number)
(lsigma =                   3.0) Lower rejection threshold
(usigma =                   3.0) Upper rejection threshold
(functio=              spline3) Fitting function for normalization spectra
(order  =                     3) Fitting function order
(sample =                     *) Sample regions
(naverag=                     1) Average or median
(niterat=                     5) Number of rejection iterations
(low_rej=                   2.0) Lower rejection sigma
(high_re=                   3.0) High upper rejection sigma
(grow   =                    0.) Rejection growing radius
(mode   =                    ql)
```

Since the aperture model for "flatcomb" has been saved to the database, one need not specify a reference model. Important parameters include (thresho = 10.), which should be a small positive number to eliminate the possibility of division by zero. Pixel values below this threshold are replaced by unity. Actually, by setting "thresho" to some factor times the read–noise, one will ensure that the pixel to pixel variations remaining in the normalized flat field will be photon statistics limited, as opposed to read–noise limited (particularly applies to aperture edges).

Set the modeling parameters (backgro = average), (pfit = fit2d), (clean = no), and (skybox = 6). This will perform the 2D extraction of Marsh (1989) with the background level being the average of the "sample" regions defined in the aperture models. Alternatively, one could use (backgro = fit), so that the background samples would be fit with the function and rejection parameters defined in the aperture model. However, the **apflatten** process is already very time intensive. Fitting the background increases the computational time even further. A boxcar smoothing of 6 pixels along the dispersion direction works well to eliminate sharp features in the background.

An important question to address for oneself is how the subtraction of background effects the scaling of the pixel to pixel variations in the resulting normalized apertures. If one performs background subtraction, the magnitude of the pixel to pixel variations will be larger than if one had not. For example, say the continuum level intensity of an aperture is 10,000 counts and the pixel to pixel variations are on the order of 1000. Direct normalization yields variations of 10%. If, however, a "background" of 5000 is subtracted off prior to normalization, the variations will be 20%. *A major problem in reducing Hamilton data is the objective determination of the true zero point at any location across the echelle format based upon sound physical principles.* The bottom line is that it is nearly impossible to normalize the illumination profiles without suffering spurious edge effects if one omits background subtraction; the cross dispersion illumination models are extremely sensitive to accurate removal of background. In other words, the routine has great difficulty cleanly modeling apertures that are effectively "overlapping".

The "clean" parameter flags the replacement of pixels deviating from the fit by more than the "low_rej" and "high_re" rejection sigma with the value of the fit itself. By setting (clean = no), all blemishes will be retained following normalization (this is the frame within which one wants to see all the uglies!). The variance weighted model is really only a Poisson model, as outlined in Appendix B. The noise model accuracy depends upon correct knowledge of the read–noise and gain. As with the various **imcombine** tasks, set (readnoi = rdnoise).

> Caution: A bug is present in V2.10.2, which effects the value of the gain one should input. Notice that this example sets (gain = 1.0), which is an incorrect gain! If one is running V2.10.2, read Appendix A; the best way to remedy this bug is to multiply the flat field frame by the gain using **imarith**, effectively scaling the pixel values to units of e^-.

Until one becomes very confident with **apflatten**, one should run it interactively. Once one knows the best overall parameters to set, one may run it non–interactively in background.

Upon ":g" the following prompt appears:

```
Flatten apertures in flatcomb?  (yes):
Fit spectra from flatcomb interactively?  (yes):
Fit spectrum for aperture 1 for flatcomb.imh interactively?  (yes):
```

Between the second and third prompts one will experience a waiting period while **apflatten** optimally extracts the spectrum of the first aperture. As illustrated in Fig. 10–1, when **apflatten** returns, one will be in **icfit** mode and can now interactively fit the dispersion direction illumination profile. If the extracted spectra is really noisy, jagged, or simply nonsense, one will need to modify the aperture models.

Upon settling on a reasonable fit, typing "q" will advance one to the next aperture. One will

need to patiently wait following each "q", while IRAF thinks. If at anytime one wishes to proceed non–interactively, type "NO" at the prompt for the next aperture.

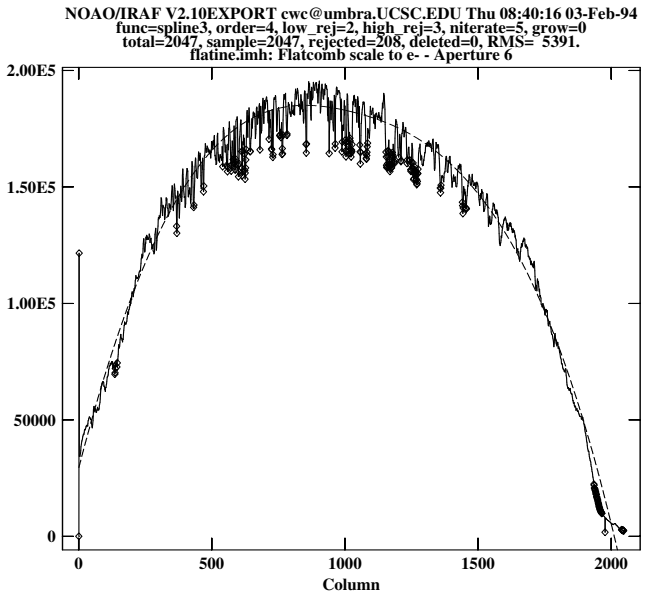


Figure 10–1. The extracted spectrum is subject to a smoothed fit to the blaze function. Normalization along the dispersion direction results from the ratio of this fitted function. Normalization of the cross dispersion profile results from the 2D optimal extraction fitting. The hope is to remove the illumination functions while retaining the pixel to pixel sensitivity (gain) variations.

10.3. Examining the Normalized Flat Field

Visual examination of the normalized flat field frame is vitally important. Are their extraction process artifacts in the normalized data? For example, is there ringing in the dispersion direction of any of the apertures. A few apertures may exhibit some awful effects.

The best way to examine the frame is with SAOimtool or XImtool using the **display** task. Visual examination will quickly reveal edge effects and other gross problems. To see more detail, run **imexamine** (also an **images tv** task). This very powerful task can produce wonderful surface, contour, and vector plots. A few useful "key" cursor commands are:

```
? Help            g Graphics cursor    h Histogram       v Vector plot
i Image cursor    c Column plot        q Quit            e Contour plot
l Line plot       s Surface plot       z Print grid      m Statistics
```

Most useful are the surface and vector plotting keys, "s" and "v". To make a surface plot, simply center the cursor on the region of interest and type "s". The size of the surface area is controlled in a separate parameter file called "simexam", which one may epar prior to executing **imexam**, or modify in real time. (Each "key" cursor command has its own parameter file; for example, the histogram parameter file is "himexam".) To make a vector plot, place the cursor at the starting pixel and type "v". Move the cursor to the ending pixel and type "v" again. To make a hard plot of the plotting window, one must first enter graphics mode by typing "g". Now do a snap ("=" or

".:snap"). To return to **imexam** image mode, type "i".

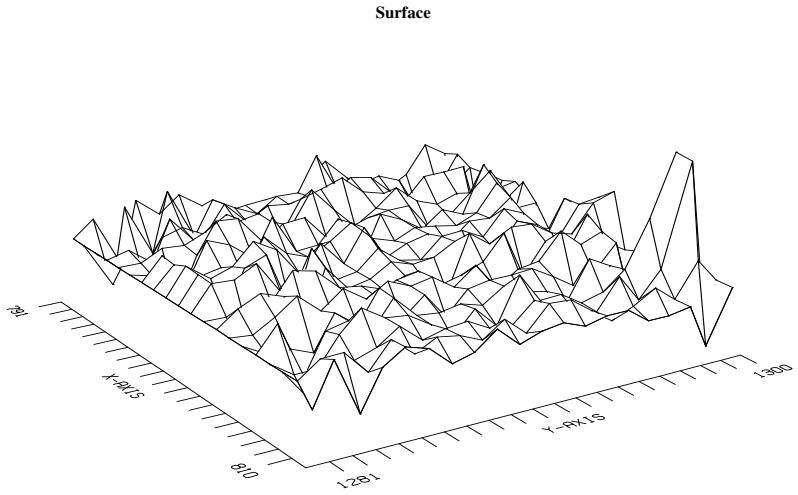


Figure 10–2. The surface plot of an arbitrary region of the normalized flat field should show pixel to pixel variations of a few percent. This is a well behaved area; there are no blemishes, only small gain variations.

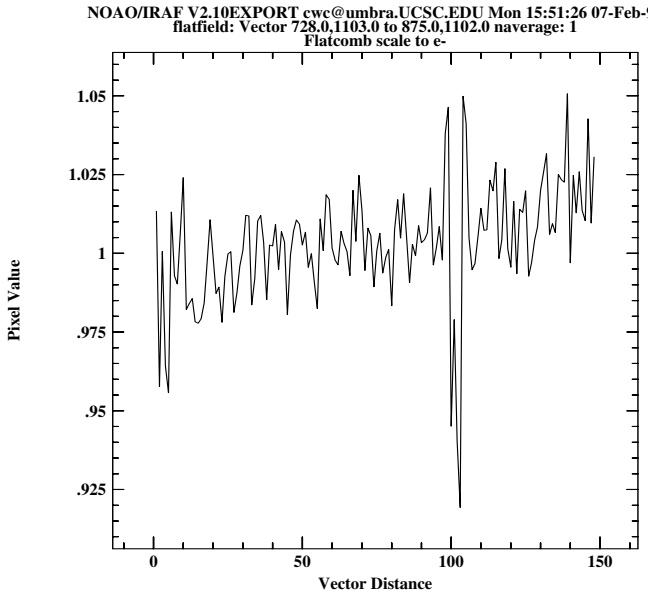


Figure 10–3. The vector plot over more than 100 pixels along the dispersion of an aperture cuts across a blemish on the detector. This blemish is probably a dust particle, which blocks several pixels in radii.

Any modeling artifacts that have amplitude on the order of the the pixel to pixel variations should be remedied. One may need to iterate on the aperture model and flat field normalization.

11. FLATTENING IMAGES AND ARCS

With a satisfactory calibration flat field in hand, it is now time to apply the multiplicative corrections to the frames with IMAGETYP = 'object'. This step is a simple second pass through **ccdproc**. The Quartz frame is now of no further use, so remove it from the @file "proclist". Also, replace "flatcomb" with the normalized flat field frame "flatfield".

> Before proceeding, check that the header card CCDMEAN of the normalized flat field frame is equal to 1.00. If it is not, use **hedit** to change the value of CCDMEAN to unity. The reason is that **ccdproc** uses the value of CCDMEAN as the mean of the flat field frame, scaling the division process by this value. IRAF V2.10.3 **apflatten** should not have this problem.

If one normalized the flat field in units of e^-, then the pixel to pixel fluctuations are in e^-. It makes no sense to divide out gain variations in e^- units into data frames that are in DN units! In this case, one needs to multiply all frames being flat fielded by the gain prior to invoking **ccdproc**.*

Epar into **ccdproc** and set the parameters for flat fielding (partial listing of pertinent parameters):

```
PACKAGE = ccdred
   TASK = ccdproc
images  =             @proclist  List of CCD images to correct
(ccdtype=                      ) CCD image type to correct
(max_cac=                     0) Maximum image caching memory (in Mbytes)
(noproc =                    no) List processing steps only?
(fixpix =                    no) Fix bad CCD lines and columns?
(oversca=                    no) Apply overscan strip correction?
(trim   =                    no) Trim the image?
(zerocor=                    no) Apply zero level correction?
(darkcor=                    no) Apply dark count correction?
(flatcor=                   yes) Apply flat field correction?
(illumco=                    no) Apply illumination correction?
(fringec=                    no) Apply fringe correction?
(readcor=                    no) Convert zero level image- readout correction?
(scancor=                    no) Convert flat field image- scan correction?
(readaxi=                  line) Read out axis (column|line)
(flat   =                      ) Flat field images
(minrepl=                 .001) Minimum flat field value
(scantyp=            shortscan) Scan type (shortscan|longscan)
```

Provided the translation table is in place, **ccdproc** will know to which images the flat corrections

* This is cheesy, I know. But it is the cleanest way to work around the V2.10.2 **apextract** bug.

apply (objects). Upon ":g", the following output will appear (partial listing):

```
thar01.bl.imh: Feb  7 17:44 Flat field image is flatfield.imh with scale=1.000
thar02.bl.imh: Feb  7 17:45 Flat field image is flatfield.imh with scale=1.000
H2Ostr.bl.imh: Feb  7 17:43 Flat field image is flatfield.imh with scale=1.000
data01.bl.imh: Feb  7 17:43 Flat field image is flatfield.imh with scale=1.000
 . . .
 . . .
 . . .
```

If the "scale" value is other than 1.0, the data have essentially been multiplied by this non–unity scale factor. To restore them, one will need to divide this scale factor back out (can you believe this trash?!?). This can be done with **imarith**. When **ccdproc** is complete, use **ccdlist** to examine the processing stages. Typing "ccdlist @proclist" results in the listing:

```
biascal.imh[2047,1600][real][zero][][OT]:bias
darkcal.imh[2047,1600][real][dark][][OTZ]:dark
flatfield.imh[2047,1600][real][flat][][OTZD]:flat
thar01.bl.imh[2047,1600][real][object][][OTZDF]:Th-Ar Lamp
thar02.bl.imh[2047,1600][real][object][][OTZDF]:Th-Ar Lamp
H2Ostr.bl.imh[2047,1600][real][object][][OTZDF]:flat
data01.bl.imh[2047,1600][real][object][][OTZDF]:Star X
data02.bl.imh[2047,1600][real][object][][OTZDF]:Star X
data03.bl.imh[2047,1600][real][object][][OTZDF]:Star X
data04.bl.imh[2047,1600][real][object][][OTZDF]:Star X
data05.bl.imh[2047,1600][real][object][][OTZDF]:Star X
data06.bl.imh[2047,1600][real][object][][OTZDF]:Star X
```

Note the presence of the processing `F` flag.

12. REMOVING SCATTERED LIGHT

The behavior of scattered light is highly complex for the Hamilton. Finding the zero point is a question of separating out the order overlap contamination from a real ubiquitous scattering throughout the instrument. Churchill and Allen (1995) show that the scattered light of the Hamilton can be represented as a two component model: (1) a global component described by a 2D polynomial, and (2) a wavelength dependent nearest neighbor component which falls off as a simple power law. Their model, HAMSCATT, is available via the Internet as a non–interactive FORTRAN module that interfaces with IRAF (Churchill 1994).

Before proceeding with scattered light removal, read §12.1. The use of **apscatter** may or may not be beneficial to ones program. In simple cases, the background can be removed during the extraction of the spectra. Otherwise, **apscatter** allows one to interactively fit functions to the *interorder minima* in the cross dispersion direction on the frame (along columns). Interactively, one fits columns of one's choosing. This is so one can settle on fitting parameters that, on average, may be applied globally to all columns. After one has settled on fiducial fit parameters, one sends **apscatter** off to automatically fit all remaining columns.

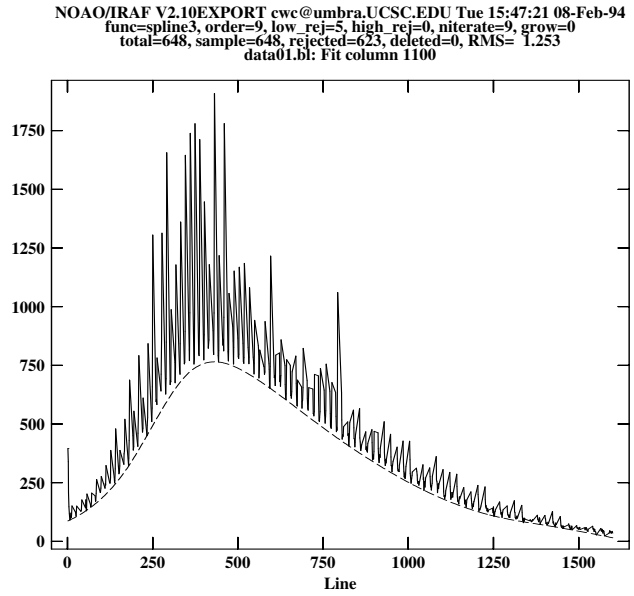


Figure 12–1. Fitting the scattered light in **apscatter** is really a simple fitting to the inter–order troughs. Fit them low. Here, ":markrej no" has been set, which removes the clutter of rejection markers. Play with the ":order" parameter until a satisfactory fit is achieved.

Since each column will have a unique 1D scattered light model, the global scattered light "surface" will be noisy from line to line. One now smoothes these column fits down lines. The smoothing procedure is identical to the column fitting procedure except that now one is fitting 1D

functions across the 1D column *fits.* After settling on fiducial fit parameters in the line direction, one again sends **apscatter** off to fit all remaining lines automatically. The result is a smoothed 2D surface that is subtracted off the data frames. In practice, **apscatter** does a below satisfactory job, simply because the Hamilton is more complex than this simple model of fitting the interorder minima. One need not remove scattered light from the Th–Ar Lamps.

Epar into **apscatter** and set the parameters:

```
PACKAGE = echelle
   TASK = apscatter
input   =             data01.bl  List of input images- subtract scatter light
output  =             data01.sc  List of output corrected images
(scatter=             scatmod01) List of scattered light images (optional)
(referen=             H2Ostr.bl) List of aperture reference images
(interac=                   yes) 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?
(subtrac=                   yes) Subtract scattered light?
(smooth =                   yes) Smooth scattered light along the dispersion?
(fitscat=                   yes) Fit scattered light interactively?
(fitsmoo=                   yes) Smooth the scattered light interactively?
(line   =                 1100.) Dispersion line
(nsum   =                     1) Number of dispersion lines to sum
(buffer =                    1.) Buffer distance from apertures
(apscat1=                      ) Fitting parameters across the dispersion
(apscat2=                      ) Fitting parameters along the dispersion
(mode   =                   ql)
```

Here, the file extension ".bl" will be replaced with ".sc", to indicate scattered light processing. However, one could leave the **apscatter** "output" parameter blank, in which case the output image would overwrite the input image. This is a matter of personal choice.

From §9, recall that the aperture model for the data is "H2Ostr.bl", so specify (referen = H2Ostr.bl). It is preferable to fit each data frame, unless they are all of the same object under *effectively identical* conditions. The accuracy of the scattered light surface is crude compared to the actual frame by frame difference between "identical" data frames. Therefore, one may wish to fit a single frame and output a model frame per the (scatter = scatmod01) parameter setting.

One must set (subtrac = yes) or the task will not execute. The param (fitsmoo = yes) should be used if one wishes to interactively "smooth" the 1D column along lines. The parameters (apscat1 =) and (apscat2 =) are default fitting parameter sets. They can be accessed by typing ":e" as such: (apscat1 = :e). The **apscat1** parameter file is

```
PACKAGE = echelle
   TASK = apscat1
(functio=                spline3) Fitting function
(order  =                      9) Order of fitting function
(sample =                      *) Sample points to use in fit
(naverag=                      1) Number of points in sample averaging
(low_rej=                     4.) Low rejection in sigma of fit
(high_re=                    0.5) High rejection in sigma of fit
(niterat=                      9) Number of rejection iterations
(grow   =                     0.) Rejection growing radius in pixels
(mode   =                      ql)
```

For the Hamilton, the "order" must be set quite high, about 9. To fit the interorder regions, tighten (high_re = 0.5) and be liberal with (low_rej = 4.). Keep the rejection iterations high. One, of course, may modify these parameters using the **icfit** commands during interactive fitting. Typing ":q" returns one to the **apscatter** epar file. Following (apscat2 = :e) one will be piped into

```
PACKAGE = echelle
   TASK = apscat2
(functio=              spline3) Fitting function
(order  =                    3) Order of fitting function
(sample =                    *) Sample points to use in fit
(naverag=                    1) Number of points in sample averaging
(low_rej=                   5.) Low rejection in sigma of fit
(high_re=                  0.5) High rejection in sigma of fit
(niterat=                    4) Number of rejection iterations
(grow   =                   0.) Rejection growing radius in pixels
(mode   =                   ql)
```

The "order" may be set quite low, about 3, since the scatter surface is a gentle parabola shape along lines. Again, tighten (high_re = 0.5) and be liberal with (low_rej = 5.). One always wants to err on the side of under estimating the scattered light, since over estimating it will result in negative values in the corrected data frame.

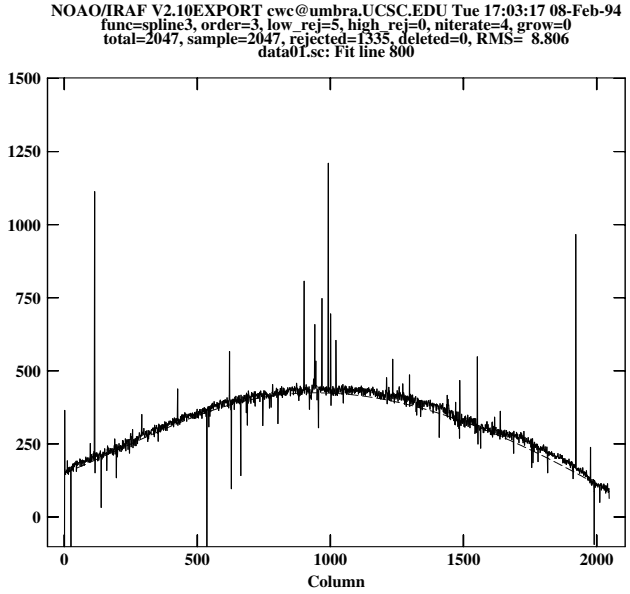


Figure 12–2. Smoothing the column to column fits is much easier, since they are usually well behaved globally. Again, fit low.

Upon returning to the **apscatter** epar file, type ":g"; following which one is prompted:

```
Write apertures for data01.bl to database  (yes):
Subtract scattered light in data01.bl?  (yes):
Fit scattered light for data01.bl interactively?  (yes):
Smooth the scattered light in data01.bl?  (yes):
Smooth the scattered light for data01.bl interactively?  (yes):
```

One will see a column "line" (set 1100 here) plotted with the fitted function drawn (see Fig. 12–1). It is recommended that one immediately type ":markrej no" followed by "f". This will remove

the rejected points markers, which are so numerous it is impossible to scrutinize the plot.

Choose fitting parameters that minimize the RMS residuals. To see the residuals, type "j". Sometimes, it is useful to perform the fits in this window. One can, of course, expand areas with the window commands "we" and "e" as with prior **icfit** steps. A "wa" will restore the full window. The "h" key returns one to the intensity verses pixel plot. Try to keep the individual residuals down to ±5% of the interorder regions. *Always underfit – positive residuals are better than negative.*

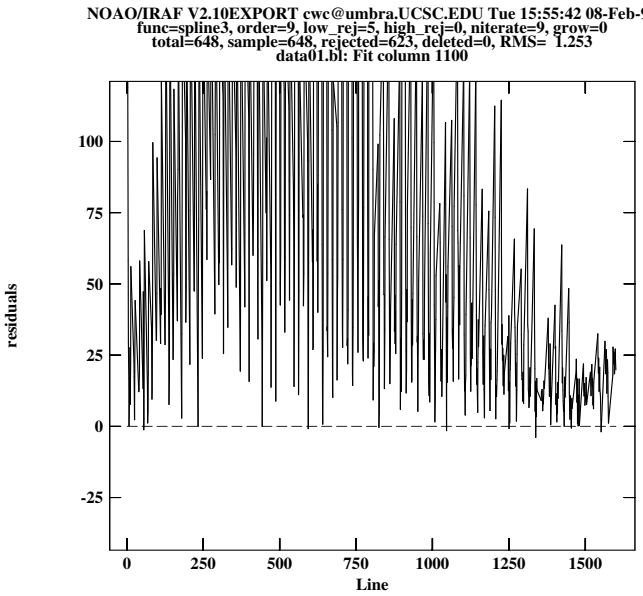


Figure 12–3. A plot of the residuals can be obtained with the "j" keystroke. Here, the residuals have been windowed for a close view. Note that the residuals are almost all positive.

One should not become "bogged down" on this first column, since one has yet to see how the fitting changes from column to column. When one has a satisfactory fit to this first column, type "q", following which one is prompted:

```
Command (quit, buffer <value>, column <value>):
```

A "q" will send **apscatter** off to fit all remaining columns with the last set of fitting parameters. Before quitting, fit a few columns near the center of the frame at about $100 - 250$ column intervals. Then fit two or so about 50 columns in from the frame edges to verify that the fitting parameters are globally satisfactory . For example, fit to some sequence like the following:

```
Command (quit, buffer <value>, column <value>): c 1250
Command (quit, buffer <value>, column <value>): c 1500
Command (quit, buffer <value>, column <value>): c 950
Command (quit, buffer <value>, column <value>): c 700
Command (quit, buffer <value>, column <value>): c 50
Command (quit, buffer <value>, column <value>): c 2000
```

Be careful, a function may work beautifully for several columns and then wildly flail on others (that which works well in the center may not work well at columns 50 or 2000, but still be a "good"

average setting). When satisfied with the cross dispersion fitting parameters, enter

```
Command (quit, buffer <value>, column <value>): q
```

and **apscatter** will automatically fit all remaining cross dispersion columns using these parameter settings. This may take $10 - 20$ minutes. When **apscatter** returns with a plot along columns (see Fig. 12–2), one will repeat the process identically, but along lines. Again, clear the rejection markers with ":markrej no".

As with columns, explore the center of the frame and then the edges. When one types "q", **apscatter** will fit all remaining lines and then exit. Visually examine the scattering model "scatmod01" with **implot** and/or **display**. Is there a high frequency of negative values? Is it smooth and reasonable? Also check the scattered light subtracted frame "data01.sc" (see Fig. 12–4).

If one trusts the model, and by visual inspection is convinced the scale and behavior of the other data frames are nearly identical to this frame, one can use **imarith** in the **images** package to apply the model to the remaining data frames by simply typing:

```
imarith data02.bl - scatmod01 data02.sc
```

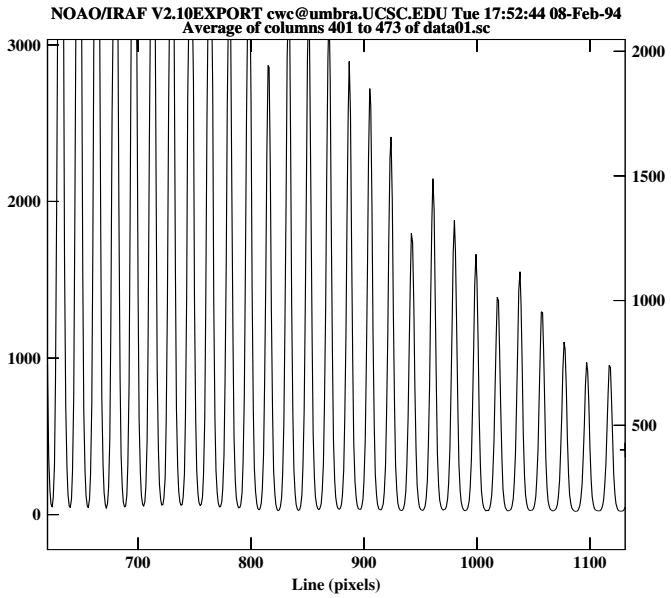


Figure 12–4. A zoom–in plot along the cross dispersion direction of various columns, or averages of columns, should illustrate that the "background" light has been removed, though using a very *ad hoc* algorithm. If one subtracts a scattered light surface from another frame and it doesn't look similar, one had better compute the model for each frame individually.

12.1. Is apscatter Useful for the Hamilton?

The main effect that **apscatter** will have on one's data is a simple shift in the zero–point. Given that the zero–point at any location in the echelle format is not known, changing it may not prove to be beneficial.

The original intent of **apscatter** is to "zero" the counts that are truly outside the edge of the projected Decker. In the case of highly separated orders, the cross dispersion information in a properly sampled aperture would be sky–data–sky, followed by a "true" interorder region and then the next aperture. Task **apscatter** is designed to remove this "true" interorder or scattered light, with the assumption that this information gives the true zero–point of the intensity projected through the Decker.

The next reduction step, variance weighted optimal extraction, may base its noise model on the counts in both the apertures and the background regions, depending if one instructs the algorithm to remove "background". Following **apscatter**, all counts will be systematically lowered, but perhaps too much so. Thus, one may skew the statistics used by the optimal extraction routine.

Alternative approaches include: (1) Do not perform **apscatter**, but subtract the background during the spectral extraction process. This approach will over estimate the zero–point; one will remove too much background. (2) Do not perform **apscatter** and do not subtract the background during the spectral extraction process. This approach effectively claims that zero intensity is the zero point, which clearly under estimates the zero–point.

For more details, see Appendix C. In §13, for purposes of example, option (1) is performed. One's own approach will, of course, be dependent upon one's scientific objective.

13. EXTRACTING THE SPECTRA

The moment of truth has arrived. Upon completing this step, one will no longer have 2D images of the apertures, but for each aperture a single 1D intensity verses pixel spectrum. The intensity at each pixel position may be computed by one of several methods, depending upon the combination one selects of the four **apall** *extraction parameters* "weights", "pfit", "clean" and "background". These are all explained in the help files for **apsum**. They are fairly well outlined in the topic files **approfiles**, **apbackground**, and **apvariance**. If one is using a large format (2000+ columns), then some form of optimal extraction is recommended, due to the large curvature of the apertures. Appendix B offers a brief outline of optimal extraction.

13.1. The Program Data

One will need to make an input @file for **apall** containing the image names for which extracted spectra are desired, namely the data and the water star. The Th–Ar Lamps will be treated separately in §13.2. Using the **files** task, type "files data*.bl H2Ostr.bl > eclist". Epar into **apall** and set parameters (partial listing):

```
PACKAGE = echelle
   TASK = apall
input   =              @eclist  List of input images
(output =                     ) List of output spectra
(format =              echelle) Extracted spectra format
(referen=            H2Ostr.bl) List of aperture reference images
(profile=                     ) List of aperture profile images
(interac=                  yes) 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 spectra?
(extras =                  yes) Extract sky, sigma, etc.?
(review =                  yes) Review extractions?
                                # EXTRACTION PARAMETERS
(backgro=                  fit) Background to subtract
(skybox =                    6) Box car smoothing length for sky
(weights=             variance) Extraction weights (none|variance)
(pfit   =                fit2d) Profile fitting type (fit1d|fit2d)
(clean  =                   no) Detect and replace bad pixels?
(saturat=                INDEF) Saturation level
(readnoi=              rdnoise) Read out noise sigma (photons)
(gain   =                   1.) Photon gain (photons/data number)
(lsigma =                   3.) Lower rejection threshold
(usigma =                   3.) Upper rejection threshold
(nsubaps=                    1) Number of subapertures per aperture
(mode   =                   ql)
```

If the null string is given for the param "output", then the input image name is used as the output *root* name with an extension, the form of which depends upon the "format" parameter. If (format = onedspec), the output aperture extractions are 1D images with names derived from the output file root name and a numeric extension given by the aperture number; i.e. "data01.bl.0001" for aperture 1 of image "data01.bl". There will be as many output images as there are apertures for each input image, all with the same root name but with different aperture extensions. If (format = echelle) or (format = multispec) then the output aperture extractions are put into a 2D image with a name formed from the output root name and the extension ".ec" or ".ms". Each line in the output image corresponds to one aperture. Thus, in this format there is one output image for each input image. These are the preferred output formats for reasons of compactness and ease of handling.

The aperture model for all the frames in @file "eclist" is (referen = H2Ostr.bl). One should set (interac = yes) and (review = yes) in order to visually inspect the extractions until one is convinced that the extraction is working properly. When one wishes to unleash the task, type "NO" at the prompt for the next aperture, following which all remaining apertures from the current image will be extracted silently. If one is really confident, typing "NO" at the "Review extracted spectra from dataXX.bl? (yes):" prompt will process all remaining images silently. This may take a considerable amount of time. Alternatively, one could verify one complete image and then extract the remaining images, running **apall** in background (interac = no). This will free one's IRAF session, allowing one to perform other work simultaneously. Obviously, set (extract = yes). If one wishes to compare the optimally extracted spectra to unweighted spectra, and record the background and sigma spectra, then one should set (extras = yes). If the "extras" parameter is set to "yes", then the above output formats become 3D. Each "band" in the third dimension contains associated information for the spectra in the first band. If optimal extractions are performed, the unweighted spectra are recorded. If background subtraction is done the background spectra are recorded. If optimal extractions are done the sigma spectrum (the estimated sigma of each spectrum pixel based on the individual variances of the pixels summed) is recorded. The order of the additional information is as outlined above. For example, an unweighted extraction with background subtraction will result in an 2–band image with band numbers 1) unweighted spectra, and 2) background spectra. A variance weighted extraction with background subtractions will result in a 4–band image with band numbers 1) variance weighted spectra, 2) unweighted spectra, 3) background spectra, and 4) sigma spectra.

Optimal extraction occurs only when (weights = variance). Recall that the "clean" parameter flags the replacement of "deviant" pixels by the illumination profile models. Note that if (clean = yes), then (weights $\equiv$ variance) – by default the spectra are optimally extracted. Unless one's spectra are virtually featureless (slowly varying), use (clean = no). "Cleaning" high resolution spectra may result in nonsense. However, set (weights = variance) and (pfit = fit2d) for the 2D optimal extraction.

For the variance model (see Appendix B), the (readnoi = rdnoise) and (gain = 1) parameters are set. Recall that IRAF V2.10.2 has a bug with the cross dispersion illumination modeling (see Appendix A). Multiply the images by the gain before executing **apall** and set (gain = 1). (As recommended in §11, the data frames may already be in e^- units.) If one chooses to remove "background", then set (backgro = fit) and (skybox = 6). For the optimal extraction, one wants the slope under the cross dispersion profile to be somewhat accurate. The functional from of the background fit, the number of rejection iterations, etc., are taken from the aperture model. A smoothing of 6 pixels along the dispersion is suggested: too little smoothing results in noisy background, while too much smoothing is not realistic. If one chooses to not remove background, then set (backgro = none).

Upon executing **apall** with ":g", one is prompted:

```
Extract aperture spectra for data01.bl?  (yes):
Review extracted spectra from data01.bl?  (yes):
Review extracted spectrum for aperture 1 from data01.bl?  (yes):
```

the latter prompt appearing after some time has elapsed. If one is reviewing the extractions, plots of the extracted spectra will appear following each prompt query (see Fig. 13–1). To advance to the next aperture, type "q". There is no interactive capability during this process. The reason one should review at least the first several extractions is to verify that the spectra match ones expectations in appearance.

If the spectra are clearly not being properly extracted, one may need to modify the aperture model. The technique to quit from **apall** is to simply type "I" for an immediate abort. Following an abort, type "flprc apall" and use **imdelete** to remove the partial output image. Now, re–execute **apall** with the (edit = yes) parameter set. One can then examine and modify the aperture models as seems appropriate to the extraction "pathology" before re–embarking on extraction.

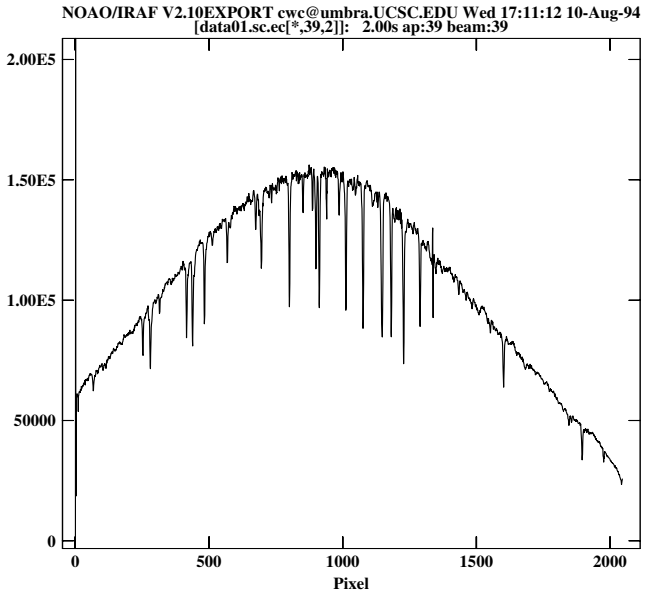


Figure 13–1. This extracted spectrum is typical of what one should see. Obvious problems would include very jagged and noisy features near the blaze profile extremes, or an overall "wavey" pattern along the dispersion direction that clearly is not a blaze profile. The latter effect may occur if (clean = yes). The blaze profile will be removed in later processing steps.

13.2. The Th–Ar Lamps

One need not apply optimal extraction to the Th–Ar Lamps. Simply set the extraction parameters as follows, leaving all other parameters unchanged:

```
input   = thar01.bl,thar02.bl  List of input images
(extract=                  yes)
(extras =                   no)
(review =                   no)
(backgro=                 none)
(weights=                 none)
(clean  =                   no)
```

With these settings, **apall** runs very quickly. Not only is optimal extraction not performed, but background subtraction is also not done. The emission features are very strong compared to any background, which is very negligible for the Th–Ar Lamp exposures. Following extraction one may

use a plotting task to view the resulting arc spectra (Fig. 13–2).

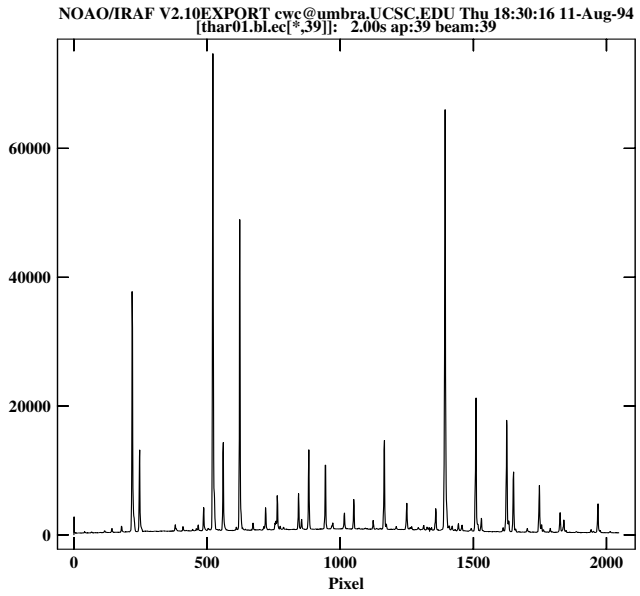


Figure 13–2. The Th–Ar Lamps do not require optimal extraction, since the signal of the emission lines is so prominent (there really is no cross dispersion profile). Apart from the details of the features, each aperture should look something like this.

13.3. Viewing the Unweighted, Background, and Sigma Spectra

One can examine multi–band images using the task **splot** in the **noao onedspec** package. This task has many powers, including spectral analysis functions. But to simply view one's extracted spectra, type "splot data01.bl.ec", and one will be prompted for the aperture number:

```
Image line/aperture to plot (0:) (1): 39
```

Type in an aperture number of interest and hit [CR], and the prompt

```
Image band to plot (1:) (1):
```

will appear. To see the optimally extracted spectrum, plot band 1. One can change apertures by using the "(" and ")" keys, to move down or up one aperture, respectively. To jump directly to an aperture by number, type "#", and one is prompted for the aperture number. To examine the unweighted extractions, type "%", and one is prompted for the band number (1=optimal, 2=unweighted, 3=background, 4=sigma). In this way, one can bounce around and study the extractions.

The background spectrum (Fig. 13–3) may contain small features of the aperture being examined and the two adjacent apertures on either side. This occurs because the Hamilton orders are so tightly

packed.

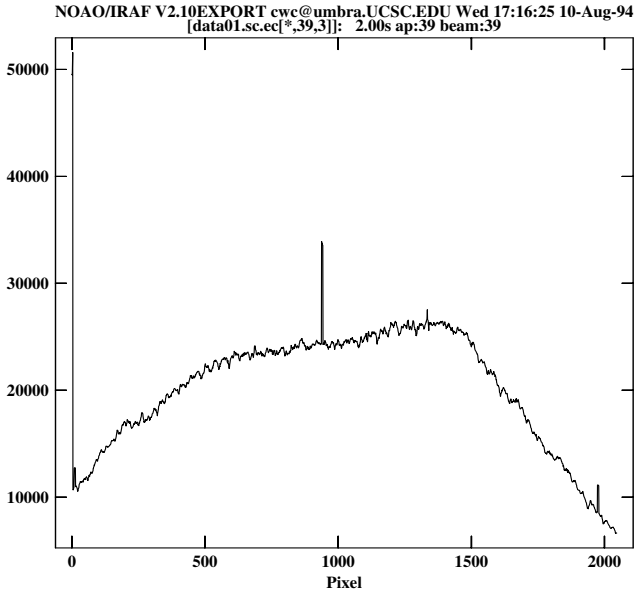


Figure 13–3. The background spectrum. Note the over–all shape of the blaze function, expected since the variance model is simple Poisson. The overlap of the Hamilton echelle orders is significant, so that the background spectra are expected to contain the strongest features of the adjacent apertures. One may be extra discretionary about "spurious" (?) features, such as the spike near pixel 1000, which will manifest as an absorption feature in the data spectrum.

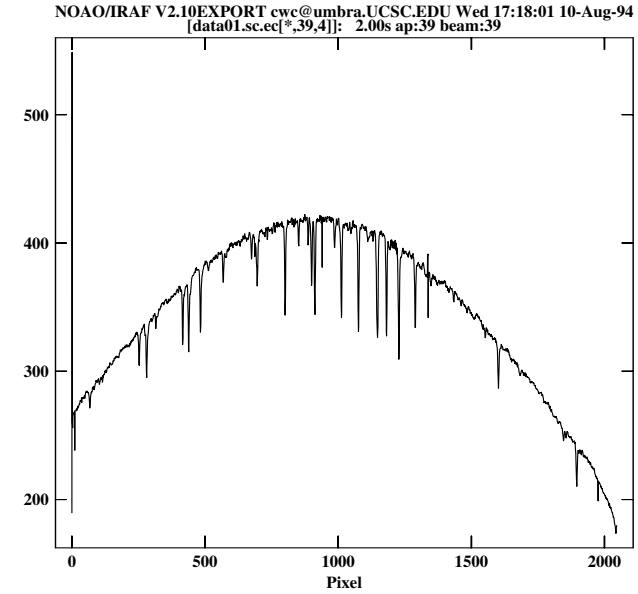


Figure 13–4. The sigma spectrum. Since the noise model is Poisson, the sigma spectra look fairly identical to the data spectra. Note the *dip* in the sigma spectrum that corresponds to the spike in the background spectrum. The same "feature" is present in the data spectrum (Fig. 13–1). Be wary that such features may be bogus.

The sigma spectrum (Fig. 13–4) should have a continuum intensity of approximately the square–root

of the data continuum. One should look for obvious features in this spectrum, indicative of deviant pixels in the resulting data spectrum. Since (clean = no), the deviant pixel has still been included in the cross dispersion sum, except that its weight in the sum has been significantly reduced (optimal extraction!).

Bouncing around and examining the spectra in **splot**, especially comparing multiple–bands, can be very instructive. At this point, one can make a more or less quantitative document of the spurious features in the spectra. In addition to the features described here, **splot** can measure such quantities as equivalent widths, perform basic spectral analysis such as line de–blending, perform arithmetic operations, including logarithmic and exponential functions, and crudely wavelength calibrate. If one knows the blaze wavelength and the dispersion for a given aperture, one may view the spectrum and compute physical quantities in wavelength space. The blaze wavelength dispersion is tabulated by Misch (1991) and Vogt (1987).

A limited but useful list of "key" commands follows:

```
b - Toggle base plot level to 0.0        ) - Go to next spectrum in image
m - Mean, RMS, snr in marked region      ( - Go to previous spectrum in image
q - Quit and exit                        # - Select new line/aperture
t - Fit continuum (see below)            % - Select new band
w - Window the graph
For 't' key: Fit the continuum with ICFIT and apply to spectrum
    / = normalize by the continuum fit
    - = subtract the continuum fit (residuals)
    f = replace spectrum by the continuum fit
    c = clean spectrum of rejected points
    n = do the fitting but leave the spectrum unchanged
    q = quit without fitting or modifying spectrum
```

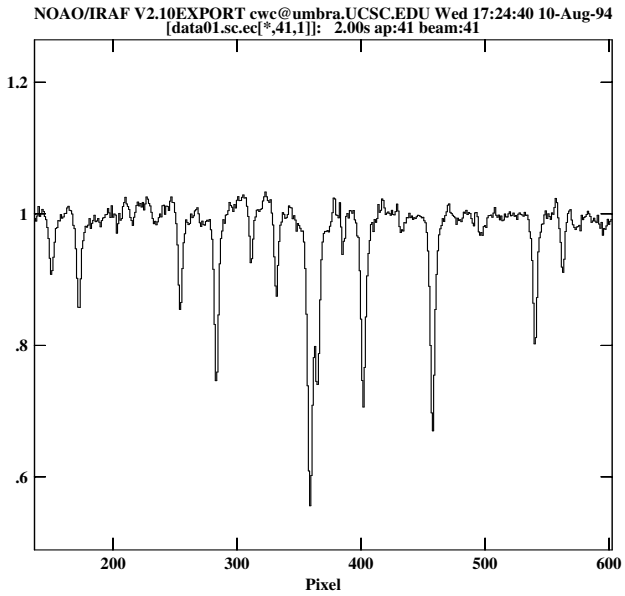


Figure 13–5. Normalized continuum fit via the "t" and "/" options. This windowed region of the aperture has been plotted using the ":hist yes" option. Plotting histogramed spectra is more revealing than "connecting the dots", giving one a better sense of the pixel sampling and the data signal–to–noise. Note the plot zero point.

If one wishes to have a preview of a continuum normalized version of the spectra, one may perform

continuum fitting in **splot**. This operation is performed on the current aperture only, and will not be retained once one changes to another aperture. Upon typing "t", one will be prompted:

```
/=normalize, -=subtract, f=fit, c=clean, n=nop, q=quit
```

Following the next keystroke, a function will be fit to the data points and the chosen operation will be executed. If one types "/", one will see the ratio of the fit. One may select only a single operation, the routine does not allow one to normalize and then decide to, say, subtract. The interactive fitting routine is (once again) **icfit**. Typing "q" returns one to the main **splot** environment.

14. WAVELENGTH CALIBRATION

Wavelength calibrating echelle formats is tedious work. The IRAF task used for echelle wavelength calibration, **ecidentfy**, is very powerful and quite capable of finding a solution with a minimum of interaction. However, one should plan to carefully manipulate the automated routines, carefully stepping toward a solution while modifying the constraints on the automation at each step. In this way, one can obtain a "science" quality calibration with confidence.

The overall process of wavelength calibration is two fold. The first step is to identify Th–Ar emission features with the help of The Hamilton Spectrograph Th–Ar Atlas (LOTR No. 73) and an on–line Thorium line list. From this, one obtains a pixel to wavelength relation (the wavelength solution) over the echelle format. The second step consists of inverting this solution and applying the dispersion correction to the pixel values. The latter step is discussed in §15. Both steps are very dependent upon mathematical modeling, sampling, and interpolation schemes and should not be taken too lightly. It is recommended that one proceed with a cautious "eye", since the data sampling may be altered (your choice) when the process is complete.

14.1. Echelle Dispersion Relation

All echelle spectrographs are governed by a basic relationship between order number and blaze wavelength (the wavelength at the mid–point along the dispersion direction of each order). The relation follows something like

$$\text{blaze } \lambda = \frac{\text{constant}}{\text{order No.}}$$

as illustrated in Fig. 14–1.

The form of the function fit explicitly includes this basic order number dependence of echelle spectra; namely the wavelength of a particular point along the dispersion direction in different orders varies as the reciprocal of the order number. Because of distortions, differing extraction paths through the 2D image, and rotations of the spectra relative to the axis of constant dispersion (i.e. aligning the orders with the image columns or lines instead of aligning the emission and absorption features) there will be residual dependencies on the extracted pixel positions and orders. These residual dependencies are fit by a 2D polynomial of arbitrary order including cross terms. Currently, the available functions are bi–dimensional chebyshev and Legendre polynomials. The fitted function is given by

$$y = \text{aperture} + \text{offset}$$

$$\lambda = f(x, y)/y$$

where y is the order number and x is the extracted pixel coordinate along the dispersion.

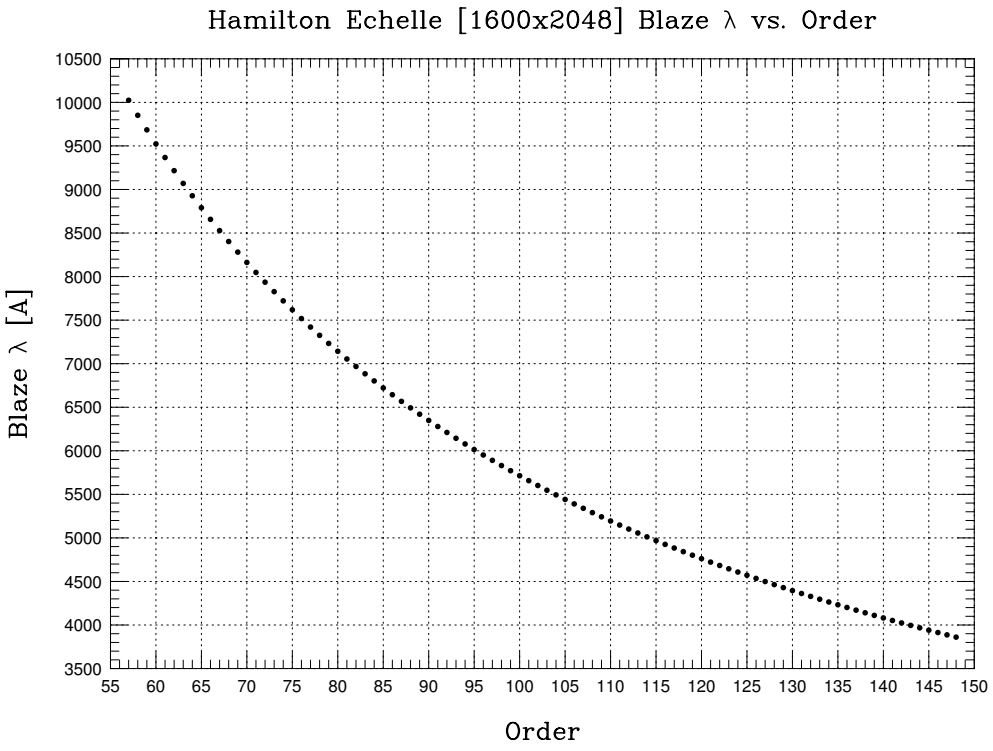


Figure 14–1. The Hamilton follows the classic relationship between wavelength and order number. For reference, Hα is located on order 87. These values were taken from Misch (1991).

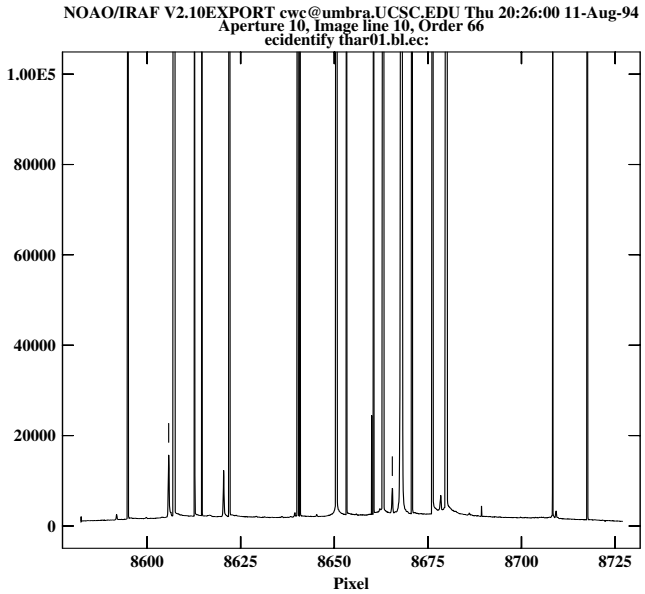


Figure 14–2. Red–ward of about 7000Å the Argon features are seriously saturated, while the Thorium features are very weak and few in number. One must capitalize on every identifiable red–ward Thorium feature.

14.2. A Calibration Cook–Book

If one has acquired the various Th–Ar Lamp exposures (as recommended in §1.6.1), one should simply add them together using **imarith** (assume that the output is called "arc.ec"). As stated in §1.6.1, the Argon features are extremely dominant red–ward of 7000Å and saturated, while the Thorium features may be barely detectable. One will find that line identification in the red will be sparse relative to the rest of the format (see Fig. 14–2).

14.2.1. The Game Plan

The wavelength solution is best done as an iterative process, a trade off between automated algorithms and manual refinement. Prior to detailing the over–all approach, one should become familiar with a few of the important parameters in the **ecidentify** task, shown below:

```
PACKAGE = echelle
   TASK = ecidentify
images  =               arc.ec  Images containing features to be identified
(databas=             database) Database in which to record feature data
(coordli= linelists$thorium.dat) User coordinate list
(match  =                 0.05) Coordinate list matching limit in user units
(maxfeat=                   50) Maximum number of features for automatic id
(zwidth =                  20.) Zoom graph width in user units
(ftype  =             emission) Feature type
(fwidth =                   4.) Feature width in pixels
(cradius=                   5.) Centering radius in pixels
(thresho=                10000.) Feature threshold for centering
(minsep =                   2.) Minimum pixel separation
(functio=            chebyshev) Coordinate function
(xorder =                    4) Order of coordinate function along dispersion
(yorder =                    4) Order of coordinate function across dispersion
(niterat=                    3) Rejection iterations
(lowreje=                   3.) Lower rejection sigma
(highrej=                   3.) Upper rejection sigma
(autowri=                   no) Automatically write to database?
(graphic=             stdgraph) Graphics output device
(cursor =                     ) Graphics cursor input
(mode   =                   ql)
```

The parameters which will be most useful (or dangerous if one neglects them!) are "match", "maxfeat", and "threshold". As one fine tunes the fitting function, one will "tighten" and "loosen" these parameters to maintain strict control over the solution.

The "match" parameter is used when one is performing automated line identifications using the on–line "coordinate list", given here as (coordli = linelists$thorium.dat). In units of wavelength, "match" defines the latitude allowed for matching features with the coordinate list entries. This parameter is also used when one is manually identifying features. During automated identifcation, "match" should be quite small so that one avoids mis–identifications. During manual identification, one should "loosen" the constraint, since coordinate list matching is under real–time scrutiny and a little "slop" actually streamlines the interactive process.

The "maxfeat" parameter is also used when one is performing automated line identifications with the coordinate list. This constraint is the maximum number of features that may be automatically matched to the coordinate list. As one's wavelength solution becomes more robust, one can "open up" this parameter following each refinement to the fit in an iterative process. Note that as the solution becomes more robust and one increases "maxfeat" to a large number (say, 1000), the "match"

parameter should subsequently be set "tighter", reducing the possibility of mis–identifications.

In order for a feature to be automatically or manually centered, the pixel intensities of the feature must exceed the value of the "threshold" parameter. At first, one will want to identify only the strongest and most obvious features. As one iterates toward a solution and increases "maxfeat", one should subsequently lower "threshold". In this way, one controls the strength of the automatically identified features, also reducing the possibility of mis–identifications.

Below is a brief overview of an iterative approach to the wavelength calibration solution:

(1) Locate an order that one identifies in the Th–Ar Atlas and manually mark a few features. Move up and down orders in steps of approximately 10, manually identifying about 3 features on each order. One may define as little as 4 features over the entire format, but full coverage as described here is better, using 20–odd features.

(2) Fit the dispersion function to these identifications. The offset will be determined, so now the orders are known and will be displayed in the plot title lines. Save your current "known" features to database by typing ":write".

(3) With "match" set to something like 0.05Å, "maxfeat" set to something like 50, and "threshold" set to something like 10000, perform automated coordinate list matching.

(4) Check the results by comparing the identifications to the Th–Ar Atlas. If all is well, ":write" your features. Then, increase "maxfeat" to 100, tighten "match" to 0.005Å, and repeat. Delete any mis–identifications.

(5) Fit the dispersion function again. This time refine the fit by examining residuals and changing the function "order" parameters. Then ":write" the results to the database.

(6) Iterate steps 3–5, increasing "maxfeat", tightening "match", lowering "threshold", and refining the fit for each successive automated coordinate list matching. Between each iteration, manually identify features in regions of the format that are sparse in coordinate matching, paying particular attention to the order dispersion ends where high order polynomials can behave poorly if not tied–down. Be sure to change the "match", "maxfeat", and "threshold" parameters appropriately. When manually identifying features, really "loosen" "match" to something like 0.5Å.

A somewhat detailed example is given in the following sub–sections. On the following pages are some useful commands. The first set are for identification and matching; the second set are for dispersion function fitting.

```
                    ECIDENTIFY CURSOR KEY SUMMARY
------------------------------------------------------------------------------
?  Help                   a  Affect all features     d  Delete feature(s)
f  Fit dispersion         j  Go to previous order    k  Go to next order
l  Match coordinate list  m  Mark feature            o  Go to specified order
p  Pan graph              q  Quit                    r  Redraw graph
w  Window graph           z  Zoom graph              .  Nearest feature
+  Next feature           -  Previous feature        I  Interrupt
------------------------------------------------------------------------------
?  Clear the screen and print menu of options
a  Apply next (c)enter or (d)elete operation to (a)ll features
d  (D)elete the  feature nearest the cursor
f  (F)it a function of pixel coordinate to the user coordinates
j  Go to the previous order
k  Go to the next order
l  Match coordinates in the coordinate (l)ist to features in the data
m  (M)ark a new feature near the cursor
o  Go to the specified (o)rder
p  (P)an to user defined window after (z)ooming on a feature
q  (Q)uit and continue with next image (also carriage return)
r  (R)edraw the graph
w  (W)indow the graph.  Use '?' to window prompt for more help.
z  (Z)oom on the feature nearest the cursor
.  Move the cursor to the feature nearest the cursor
+  Move the cursor to the next feature
-  Move the cursor to the previous feature
I  Interrupt task immediately.  Database information is not saved.
------------------------------------------------------------------------------

                   ECIDENTIFY COLON COMMAND SUMMARY
------------------------------------------------------------------------------
:show [file]              :coordlist [file]          :cradius [value]
:threshold [value]        :database [file]           :match [value]
:maxfeatures [value]      :minsep [value]            :zwidth [value]
------------------------------------------------------------------------------
The parameters are listed or set with the following commands which may be
abbreviated.  To list the value of a parameter type the command alone.
:show file              Show the values of all the parameters
:coordlist file         Coordinate list file
:cradius value          Centering radius in pixels
:threshold value        Detection threshold for feature centering
:match value            Coordinate list matching distance
:maxfeatures value      Maximum number of features automatically found
:minsep value           Minimum separation allowed between features
:zwidth value           Zoom width in user units
------------------------------------------------------------------------------
```

```
              ECHELLE DISPERSION FUNCTION FITTING KEYS
------------------------------------------------------------------------
                         CURSOR KEY SUMMARY
------------------------------------------------------------------------
?  Help             c  Print coordinates             d  Delete point
f  Fit dispersion   o  Fit with fixed order offset   q  Quit
r  Redraw graph     u  Undelete point                w  Window graph
x  Set ordinate     y  Set abscissa                  I  Interrupt
------------------------------------------------------------------------
?  Print this list of cursor keys
c  Print cursor coordinates
d  Delete the nearest undeleted point to the cursor
f  Fit dispersion function including determining the order offset
o  Fit dispersion function with the order offset fixed
q  Quit and return to the spectrum display
r  Redraw the graph
u  Undelete the nearest deleted point to the cursor
w  Window the graph (type ? to the window prompt for more help)
x  Set the quantity plotted along the ordinate (x axis)
y  Set the quantity plotted along the abscissa (y axis)
I  Interrupt the task immediately
------------------------------------------------------------------------

                         COLON COMMAND SUMMARY
------------------------------------------------------------------------
:show                :function [value]     :highreject [value]
:lowreject [value]   :niterate [value]     :xorder [value]
:yorder [value]
------------------------------------------------------------------------
:show                Print current function and orders
:function [value]    Print or set the function type (chebyshev|legendre)
:highreject [value]  Print or set high rejection limit
:lowreject [value]   Print or set high rejection limit
:niterate [value]    Print or set number of rejection iterations
:xorder [value]      Print or set the order for the dispersion dependence
:yorder [value]      Print or set the order for the echelle order dependence
------------------------------------------------------------------------
```

14.2.2. A First Fit

Upon executing **ecidentify**, one will see a plot or aperture 1 (simliar to Fig. 14–2); there will be no prompt. For consistency the orders are always identified by their aperture numbers in this task and all other tasks. These are the identifications assigned when extracting the orders using the **apextract** package tasks. Initially the orders are the same as the aperture numbers, but after fitting a dispersion function the true order numbers will be determined. This information is indicated in the graph titles, but selection of an order to be graphed with "o" and status line information is always in terms of the aperture number.

To move up one order at a time use the "k" key. To move down one order at a time, use the "j" key. To plot a specific order, type "o" and type in the order number (actually aperture!) at the prompt. Find a representative order near the center of the format (see Fig. 13–2) as a starting point and compare it to the Th–Ar Atlas. The order numbers are given in the graph titles of the Atlas, so one should not have too much difficulty here, though this is the critical step. One's own aperture numbers may not correspond to the *aperture* numbers in the Atlas. If they do not, the offset between one's own apertures and the Atlas apertures will be constant, so all subsequent order identifications are straight forward. If one moves up 10 orders in **ecidentify**, one moves up 10 orders in the Atlas!

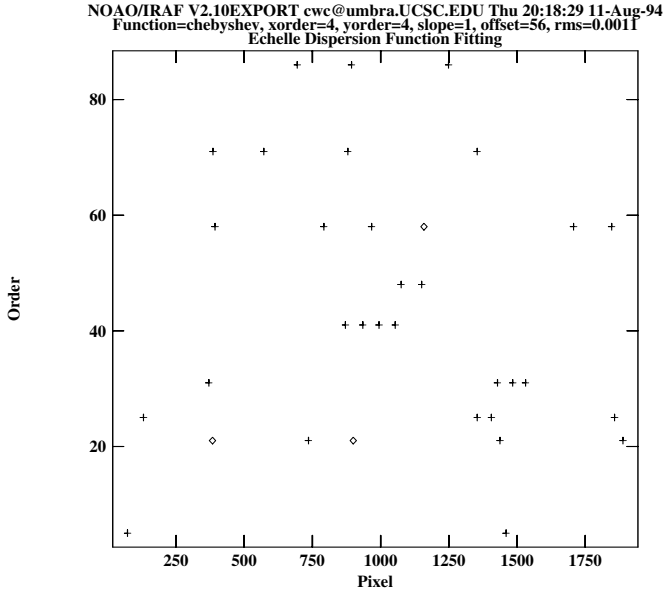


Figure 14–3. Following the "f" keystroke, one may choose from a selection of plots to examine the fitting characteristics. Here is a plot of echelle order verses pixel, showing the location of identified lines over the format. The diamond data points are lines that have been sigma rejected from the 2D fit.

To manually identify a feature, set the cursor within the feature and strike the "m" key. Match features that are above the "threshold" setting, or one will hear a beep and the feature will not be centered. Often, one may hear a beep due to the cursor not being properly centered on the feature. If so, move the cursor a bit to the right of the feature and try again. Following the "m" keystroke, a small vertical bar appears over the feature, and one is prompted with:

```
31     988.40  988.40 (     INDEF):
```

which are the aperture number, pixel number (column), dispersion fit coordinate, and the coordinate list match default, respectively. Since no fit has been performed at this point, the dispersion fit

coordinate is simply a repeat of the pixel number, and the coordinate list default is "INDEF". Type in the feature identification from the Atlas and hit [CR]. To delete a feature, place the cursor near the feature and type "d" followed by "r". The "d" key deletes the feaure nearest the cursor but does not update the plot, which is done with "r".

If a minimum of four features over at least two orders have been identified, a dispersion function fit may be performed. However, more features are preferable to determine variations in the dispersion as a function of position and order. By manually idenitifying features every 10th order or so, good coverage can be obtained for the first fit (see Fig14–3).

To fit the dispersion, type "f". After a moment, one will see a plot of residuals verses pixel, the latter being the column number in which the identified features are located. In fitting mode, one may examine the current dispersion solution using several plots. To control the ordinate, type "x". One will see:

```
Ordinate - (p)ixel, (o)rder, (w)avelength, (r)esidual, (v)elocity:
```

and likewise, to control the abscissa, type "y". One will see:

```
Abscissa - (p)ixel, (o)rder, (w)avelength, (r)esidual, (v)elocity:
```

for example, to examine the format coverage of the current identifications use "xp" and "yo", as was done to obtain Fig. 14–3. When one has obtained a final dispersion solution, this plot will be scattered with points (see Fig. 14–7). At this point, don't attempt to refine the residuals by changing ":xorder" and ":yorder", though upon exiting fit mode with "q", one may wish to examine any features with large residuals against the Th–Ar Atlas to check for mis–identifications.

14.2.3. Matching Coordinates: First Pass

To better the dispersion fit, loosen "match" to about 0.05Å and manually fit some more lines using "m". Now that a dispersion function is in place, one will be prompted with:

```
31    1072.04  6569.6295 ( 6569.6322):
```

Note the discrepancy between the dispersion fit coordinate (3rd entry) and the coordinate list match in "()". Had "match" been smaller than the difference of these two values, the default would have been "INDEF" and one would be required to type in the feature identification. With "match" opened up, many features will be matched so that one may simply hit [CR]. This saves considerable time. During this process, almost every feature will match with the coordinate list. One should cross check every identification with the Th–Ar Atlas during manual identification.

If all is well, tighten "match" somewhere between 0.0005 and 0.001 Å, kick up "maxfeat" to 100, and lower "threshold" to 5000. Automatically match features by typing "l", one will see:

```
Searching coordinate list ...
```

When **ecidentify** returns, type "f" and examine the order vs pixel plot to examine the results, as shown in Fig. 14–4. Note the format coverage and the total number of identified features, as well as

the number of sigma rejected features (diamond markers).

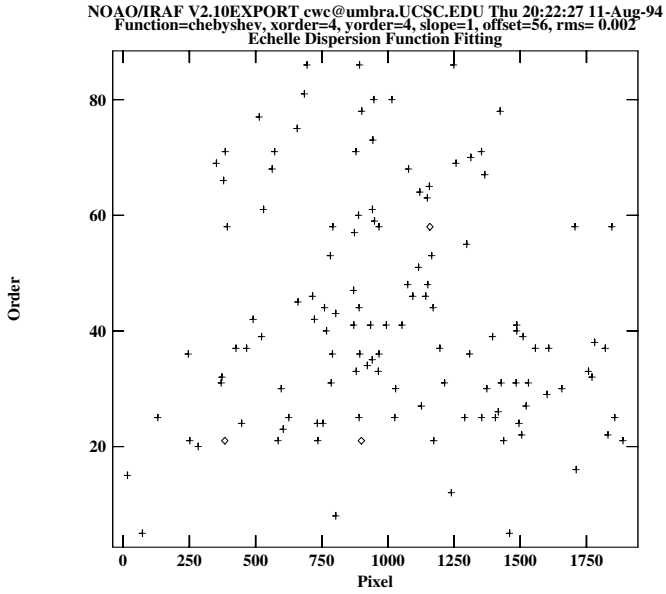


Figure 14–4. Same as Fig. 14–2 following the "l" keystroke (automated coordinate matching) with ":maxfeat=100". Note how the format coverage has increased, though is still very limited. In particular, the order ends need to be "nailed down", perhaps manually.

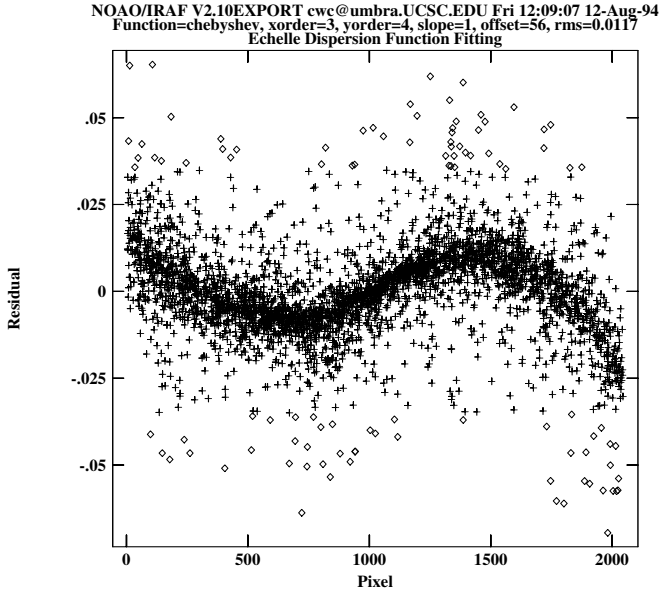


Figure 14–5. The residuals clearly show structure across columns, indicating that the ":xorder" is set too low. Here, ":xorder=3". Setting ":xorder=4" mostly removed the structure, though smaller amplitude higher frequency wave–like structure was still present. At some level, such structure cannot be completely eliminated.

14.2.4. Tuning the Fit

Usually, interactive fitting is performed with the goal of minimizing the fitting residuals, including any global trends and patterns in the residuals. Examine the residuals verses pixel and residuals verses wavelength plots for obvious trends, such as wave–like behaviour and holes in wavelength coverage (see Fig. 14–5). Using the ":xorder" and ":yorder" commands, one can reduce any obvious structure in the residuals. During the iterative process, use the "o" key when fitting. This fitting option holds the "offset" term constant (though one is prompted with the default), and is much faster.

By typing ":show", one may view all the current fitting parameters. As with all fitting, be sure that the number of rejection iterations (":niterate") is at least a few. After exiting from fitting with "q", save these intermediate results to the database by typing ":write".

14.2.5. Matching Coordinates: Second Pass, Third Pass...

Now one may return to feature identification with the goal of nailing down the order ends, removing any gross mis–identifications, and uniformly filling out the format coverage with identified features. One should manually identify the features near the order ends, since the fit may be somewhat wild and automated matching may mis–identify features. This would compound bad results on the order ends, since the fit would be tied–down with incorrect identifications (the blind leading the blind). Again, when manually identifying, loosen "match" and carefully verify each default coordinate list match with the Th–Ar Atlas.

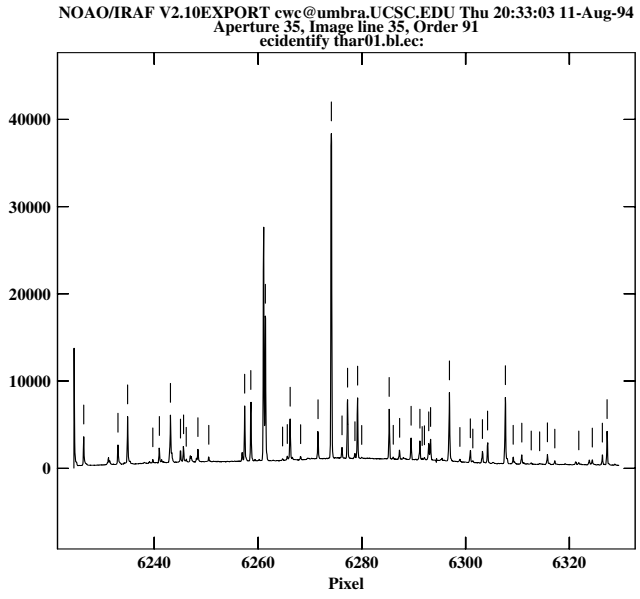


Figure 14–6. Once one has added thousands of line identifications, each order should have tens of identified features. Inspect each order and maually insert lines where need be, particularly the order ends.

Once the order ends are secure, one may tighten "match" and perform automatic identification with "l". This time, crank down on "match" to no greater than 0.0005Å. Also, kick up "maxfeat" to 500, 1000, and 3000 in successive steps through this process. Follow each iteration with a fit, checking that the fit does not degrade and that the format coverage is becoming uniformly sampled

(Fig. 14–7). One may need to pay extra attention to features redward of 7500Å. They will not be matched unless "threshold" is reduced to $500 - 1000$. Zoom in on these orders and be sure of the proper value, being sure that "threshold" is comfortably above the "continuum"'.

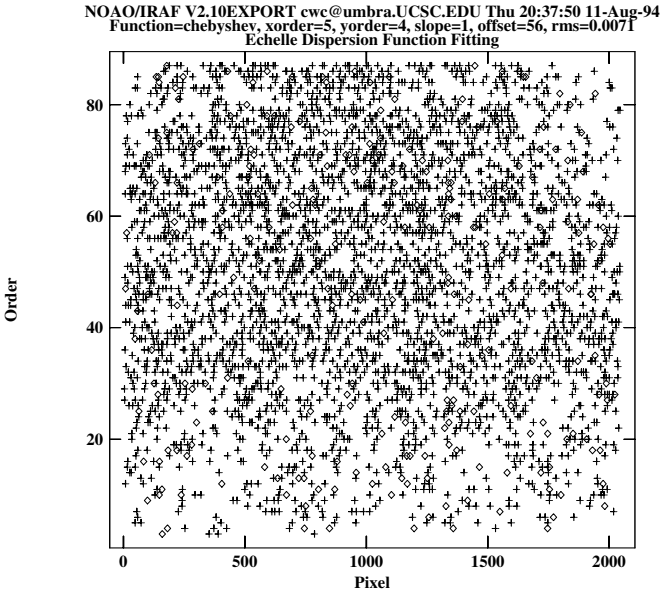


Figure 14–7. Same as Figs. 14–3 and 14–4 following "l" with :maxfeat=3000. Note the full format coverage, except toward the very red (lower order numbers).

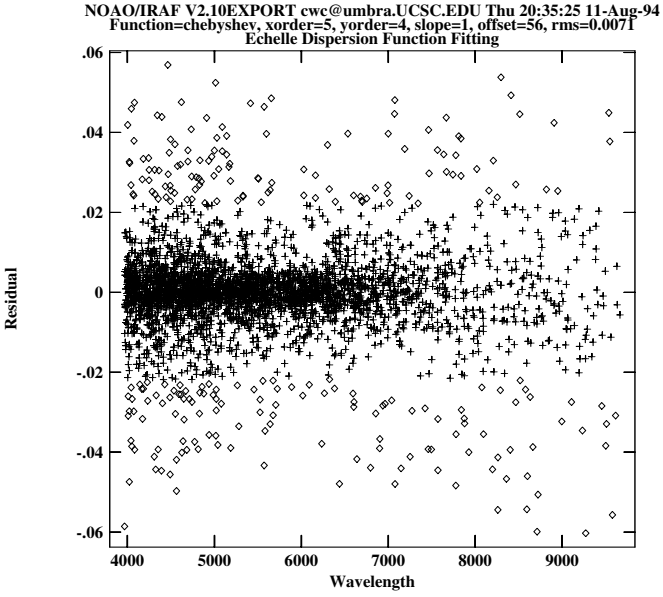


Figure 14-8. A plot or fitting residuals verses wavelength. One modifies the fitting function until the residuals are at an acceptable value. In particular, one is attempting to remove any wave–like structure in the residuals as a function of both pixel position (both columns x and lines y) and wavelength (shown here).

14.2.6. Fine Tuning the Fit

At this point one should have a fairly good idea on how to proceed. The main concerns have been emphasized above. During the final fitting attempt, one should simply bounce around the "yr"–"xp" plot, minimizing the RMS residuals and residual wave patterns along columns with ":xorder", and the "yr"–"xo" plot, minimizing the RMS residuals and residual wave patterns along orders with ":yorder". Further, one can also check residual patterns with the "yr"–"xw" plot, looking for oscillating patterns over certain wavelength regimes (see Fig. 14–8). The final fit here was an "xorder=5" "yorder=4" chebyshev, which is typical of the 2000+ column format.

14.3. The Database Record

Like the aperture solution, the wavelength solution is recorded in the database subdirectory in a text file. The file names are prefixed with "ec" followed by the input image name. The database text file consist of a number of records, each record beginning with a line starting with the keyword "begin". The rest of the line is the record identifier. The lines following the record identifier contain the feature information and dispersion function coefficients.

```
# Thu 20:19:09 11-Aug-94
begin   ecidentify arc.ec
        id      arc.ec
        task    ecidentify
        image   arc.ec
        features        34
                  5   61    72.51   9291.5312   9291.5313   4.0  1  1
                  5   61  1459.46   9399.0897   9399.0891   4.0  1  1
                 21   77   383.70   7380.4153   7380.4263   4.0  1  0
                 21   77   735.24   7402.2528   7402.2521   4.0  1  1
                 21   77   899.73   7412.3453   7412.3368   4.0  1  0
                 . . .
                 . . .
                 . . .
                 86  142   694.44   4012.4955   4012.4952   4.0  1  1
                 86  142   893.04   4019.1284   4019.1289   4.0  1  1
                 86  142  1248.80   4030.8429   4030.8424   4.0  1  1
        offset  56
        niterate 5
        lowreject 3.
        highreject 3.
        coefficients    24
        . . .
        . . .
        . . .
```

The database **ecidentify** files can be extremely long.

15. ATTACH AND APPLY THE DISPERSION SOLUTION

The process of applying the wavelength calibration to the object spectra is two stepped. The first is to simply place information in the header of the object spectra as to how and which dispersion solutions (if more than one is to be used) are to be applied, including options such as averaging and interpolating between solutions. The second is to actually perform the dispersion correction, which results in the wavelength calibrated spectra.

15.1. Attach the Dispersion Solution

Here, we perform the first step. The task **refspectra** of the **noao onedspec** package allows one to define which reference spectra are to be used in the calculation of the dispersion "correction" for the object spectra. The assignment of reference spectra to object spectra may be a complex step because of the number of spectra, the use of many distinct apertures, and different modes of observing such as interspersed arc calibration spectra or just a single calibration for a night. This task provides a number of methods to cover many of the common cases. The example given below will be the simple application of a single reference spectrum taken at the beginning of the night. Recall that the Hamilton is very stable on time scales of 24 hours. See the **refspectra** help file on details regarding more complex applications. Epar into **refspectra** and set the parameters:

```
PACKAGE = echelle
   TASK = refspectra
input   =              @eclist  List of input spectra
(referen=               arc.ec) List of reference spectra
(apertur=                     ) Input aperture selection list
(refaps =                     ) Reference aperture selection list
(ignorea=                  yes) Ignore input and reference apertures?
(select =                match) Selection method for reference spectra
(sort   =                     ) Sort key
(group  =                     ) Group key
(time   =                   no) Is sort key a time?
(timewra=                  17.) Time wrap point for time sorting
(overrid=                   no) Override previous assignments?
(confirm=                  yes) Confirm reference spectrum assignments?
(assign =                  yes) Assign reference spectra to the input spectr
(logfile=       STDOUT,logfile) List of logfiles
(verbose=                  yes) Verbose log output?
answer  =                  yes  Accept assignment?
(mode   =                   ql)
```

The param "select" is central to the options available for this task. It is not very relevant with a single reference spectrum. Note that "verbose" output is not very useful for the present application of **refspec**. The output will be something like (provided one responds with [CR] at each prompt):

```
[H2Ostr.ec] refspec1='arc.ec'   Accept assignment? (no|yes|YES) (yes):
[data01.ec] refspec1='arc.ec'   Accept assignment? (no|yes|YES) (yes):
```

```
[data02.ec] refspec1='arc.ec'   Accept assignment? (no|yes|YES) (yes):
[data03.ec] refspec1='arc.ec'   Accept assignment? (no|yes|YES) (yes):
[data04.ec] refspec1='arc.ec'   Accept assignment? (no|yes|YES) (yes):
[data05.ec] refspec1='arc.ec'   Accept assignment? (no|yes|YES) (yes):
[data06.ec] refspec1='arc.ec'   Accept assignment? (no|yes|YES) (yes):
```

For this example, **refpsectra** has simply inserted the header card REFSPEC1 = 'arc.ec' into the input images. At this point, one may change the reference spectrum assignment if one needs to for some reason. This is accomplished using the "overrid" parameter. Actually, given this simple application of **refspectra**, where we simply attached a single reference spectrum, we could have skipped this step entirely and simply use the "table" parameter in the task **dispcor** (see below).

15.2. Apply the Dispersion Correction

The wavelength solution of **ecidentify** is of the form describing the wavelength for a given pixel location, not a pixel location for a given wavelength. The solution must be inverted. The inversion process involves linear interpolation between contiguous wavelength values to obtain pixel values at equal intervals of wavelength. Once the solution is inverted, one may "linearize" the data, which really means re–bin the data by either (1) interpolating between equal intervals of wavelength with a choice of interpolation function, or (2) re–binning by partial pixel summation. The other option is to leave the data in their original state (see the "lineari" parameter in **dispcor**). More details can be obtained by reading the **onedspec** package help file.

15.2.1. Setting The Interpolation Type

If one chooses to linearize the data, the input spectrum will be interpolated to obtain new values for each output pixel. The default interpolation function for re–mapping is a 5th order polynomial. The choice of interpolation type is made with the **onedspec** package parameter "interp". The help entry for "interp" follows:

```
interp = 'poly5' (nearest|linear|poly3|poly5|spline3|sinc)
    Spectrum interpolation type used  when  spectra  are  resampled.
    The choices are:
            nearest - nearest neighbor
             linear - linear
              poly3 - 3rd order polynomial
              poly5 - 5th order polynomial  (default)
            spline3 - cubic spline
               sinc - sinc function
```

The choice of interpolation type depends on the type of data, smooth verses strong, sharp, or under sampled features, and the issues of one's program. The "nearest" and "linear" interpolation are crude and simple but they avoid "ringing" near sharp features. The polynomial interpolations are smoother but introduce notable ringing near sharp features. They are, unlike the "sinc", localized. "Sinc" interpolation approximates applying a phase shift to the Fourier transform of the spectrum. If there are no under sampled narrow features it is by far the best choice, but when there are such features the contamination of the spectrum by ringing is much more severe than with other interpolation types. If one wishes to change the default settings, epar into the **onedspec** package

and set the parameters:

```
PACKAGE = noao
   TASK = onedspec
(observa=          observatory) Observatory for data
(caldir =                     ) Standard star calibration directory
(interp =                poly5) Interpolation type
(dispaxi=                    1) Image axis for 2D images
(nsum   =                    1) Number of lines/columns to sum for 2D images
(records=                     ) Record number extensions
(version= ONEDSPEC V3: January 1992)
(mode   =                   ql)
($nargs =                    0)
```

Remember that these apply to all tasks which might need to interpolate spectra in the **onedspec** and associated packages. For a more detailed discussion of interpolation types see the **onedspec** help file (one must have loaded **onedspec**, then type "help package").

15.2.2. The Dispersion Correction

The task **dispcor** is used to apply the dispersion correction. If no output list is specified then the output spectrum will replace the input spectrum. This task has many ways of defining the wavelength coordinates of the output spectra, ranging from interactively defining them to using a reference table or image. For images that have attached reference spectra (from **refspectra**) the dispersion function is specified by one or both of the reference spectrum image header cards REFSPEC1 or REFSPEC2 which name the calibration spectra with echelle dispersion functions placed in the database subdirectory by **ecidentify**. Epar into **dispcor** and set the parameters:

```
PACKAGE = echelle
   TASK = dispcor
input   =              @eclist  List of input spectra
output  =                       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?
(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=                dclog) Log file
(mode   =                   ql)
```

By setting (databas = database) and leaving (table =) the dispersion solutions are taken from the database files which are named in the header cards REFSPEC1 and REFSPEC2 of the input list spectra in "eclist". If (lineari = yes), then the spectra are interpolated to a linear wavelength scale and the dispersion coordinate system in the header is set appropriately. The param (flux =

yes) is set to conserve the total flux during interpolation – the output pixel values (the intensities) are obtained by integrating the interpolation function across the wavelength limits of the output pixel. If (flux = no), the output spectrum is interpolated from the input spectrum at each output wavelength coordinate. If one wishes to verify and possibly change the defaults assigned, either globally or for individual apertures, the "confirm" flag may be set. The resulting linear wavelength coordinate systems for each aperture are defined by a starting wavelength, an ending wavelength, a wavelength interval per pixel, and the number of pixels, designated "w1", "w2", "dw', and "nw", respectively. If (lineari = no), then the spectra are not interpolated and the "flux" parameter does not apply. If one wishes to have a copy of the wavelength coordinate assignments for the apertures, define a "logfile".

The (lineari = yes) (verbose = yes) output will look something like:

```
data01.ec: ap = 1, w1 = 9937.405, w2 = 10104.69, dw = 0.081844, nw = 2047
data01.ec: ap = 2, w1 = 9766.113, w2 =  9930.53, dw = 0.080439, nw = 2047
data01.ec: ap = 3, w1 = 9600.616, w2 = 9762.258, dw = 0.079081, nw = 2047
data01.ec: ap = 4, w1 = 9440.627, w2 = 9599.586, dw = 0.077769, nw = 2047
                                  ...
                                  ...
                                  ...
data01.ec: ap = 90, w1 = 3879.589, w2 = 3945.131, dw = 0.032066, nw = 2047
data01.ec: ap = 91, w1 = 3853.195, w2 = 3918.295, dw = 0.031849, nw = 2047
data01.ec: ap = 92, w1 = 3827.159, w2 = 3891.821, dw = 0.031635, nw = 2047
```

for each spectrum in the input list, **dispcor** will go away and think for a few minutes between each spectrum, depending upon the interpolation type ("sinc" takes longer).

15.3. Looking at Calibrated Spectra

At this stage, for the first time, one can now plot one's spectra in wavelength space, instead of pixel space. Though the blaze function still dominates the continuum profile, one may wish to inspect the spectra at this stage. Most of the following discussion may be more interesting to one following continuum normalization. In the **onedspec** package there are three useful tasks for looking at wavelength calibrated spectra:

```
   bplot - Batch plots of spectra using
specplot - Stack and plot multiple spectra
   splot - Interactive spectral plot/analysis
```

15.3.1. bplot

The **bplot** task is a script that drives **splot**; it is easy to use and very versatile. If one wishes to get fancy (or save paper!), one can redirect the output to a "metafile" and use tasks in the **plot** package to display the metafile contents. For example, **gkimosaic** makes multiple plots per page. A few possibilities with **bplot** include:

(1) Plot each aperture of the multi–aperture spectra of "data01.ec" to the screen one at a time at about 2 second intervals (non–interactive). This is a useful quick look–see.
command: `bplot data01.ec aper= ''`

(2) Plot each aperture of the multi–aperture spectra in the @file "eclist" on the default plotter and run in the background.
command: `bplot @ecfile aper='' graphics=stdplot &`

(3) Produce plots with four spectra per page (default). First, create the metafile "ecfile.mc", then pipe the metafile to the printer.
command: `bplot @ecfile ... >G ecfile.mc | gkimosiac ecfile.mc dev=stdplot`

15.3.2. splot

The **splot** task is a powerful "machine", as was briefly discussed in §13.3 (see Figs. 13–2 through 13–5). Recall that one can perform complicated operations on the data, including continuum normalization. In fact, one could complete the reduction within this task alone.

15.3.3. specplot

The **specplot** task has unique options, including plotting any order of a given data frame next to orders from other data frames and in specified sizes, orientations, and with labels! This task plots multiple spectra with provisions for scaling, vertically separating, and horizontally shifting them. One can view all orders stacked upon one another, or one can string out the entire spectrum with overlapping wavelength coverage from adjacent orders! The latter is useful for comparing identical wavelength coverage at two distinct data points. The key parameter is "fractio". If set to '0.', **specplot** simply plots the entire wavelength range without offsetting the orders vertically (very compressed in wavelength).

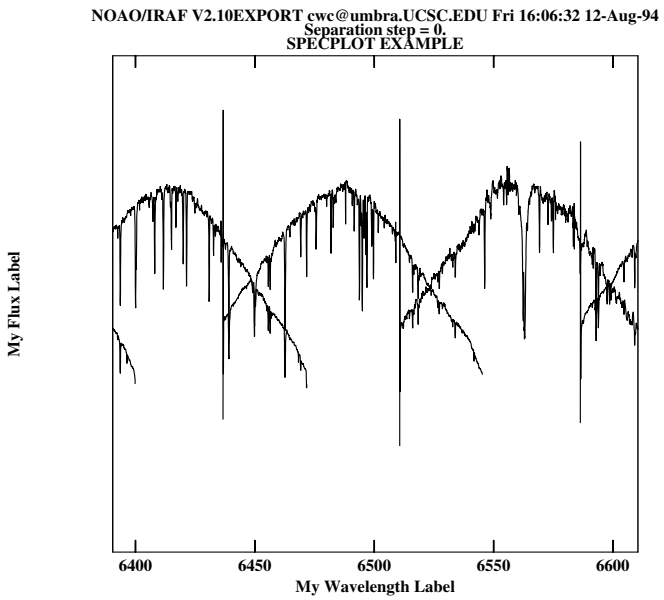


Figure 15–1. Three orders are plotted with spectral overlap clearly shown. Note the excellent double coverage of the line at 6450Å. The large absorption feature at 6553Å is Hα (no kidding!). The repeating spike is a bad column.

16. NORMALIZE THE CONTINUUM

The spectra still have the blaze profile of the grating. This is removed by task **continuum** in the **noao onedspec** package. This step in the reduction is very important to perform interactively on each and every scientifically interesting aperture of each and every data frame. The primary reason is that the **continuum** task is not very versatile and the resulting continuum may have medium frequency ripples. This task needs improvement, such as allowing the user to specify the positions of the spline "knots".

The **continuum** task is equipped to record which apertures of which spectra have been normalized and which have not, by placing the header card SFIT into processed frames. Thus, one may work through one's "eclist", exit **continuum** and start up at a later date without any difficulties, even if one left off at the 3rd aperture of the 5th image.

Epar into **continuum** and set the parameters:

```
PACKAGE = echelle
   TASK = continuum
input   =              @eclist  Input images
output  =                       Output images
(lines  =                    *) Image lines to be fit
(type   =                ratio) Type of output
(replace=                   no) Replace rejected points by fit?
(wavesca=                  yes) Scale the X axis with wavelength?
(logscal=                   no) Take the log (base 10) of both axes?
(overrid=                   no) Override previously fit lines?
(listonl=                   no) List fit but don't modify any images?
(logfile=              logfile) List of log files
(interac=                  yes) Set fitting parameters interactively?
(sample =                    *) Sample points to use in fit
(naverag=                    1) Number of points in sample averaging
(functio=              spline3) Fitting function
(order  =                    4) Order of fitting function
(low_rej=                  2.0) Low rejection in sigma of fit
(high_re=                  2.0) High rejection in sigma of fit
(niterat=                    5) Number of rejection iterations
(grow   =                   1.) Rejection growing radius in pixels
(markrej=                   no) Mark rejected points?
(graphic=             stdgraph) Graphics output device
(cursor =                    ) Graphics cursor input
ask     =                  yes
(mode   =                   ql)
```

If one does not specify an output list, the output images will retain the input image names. The (lines = *) param instructs **continuum** to fit all orders. Setting (type = ratio) results in normalization set to unity. Set param (replace = no). One does not wish to replace rejected points by the fit, the majority of which will be spectral features! It is good to keep a logfile if one ever needs to examine one's work history.

The important fitting default parameters are (low_rej = 2.0) and (high_re = 2.0), which should be set to reject spectral absorption features (if one makes extensive use of the ":sample" command, one may need to modify these settings). Set (niterat = 5), which is enough iterations to converge on the fit with no noticeable slowing of the routine. There will be so many rejected points that if (markrej = yes), the plot will be dominated by cross–hatches that all but block the view of the fit. Set (markrej = no). This param may be toggled (colon command) during interactive fitting whenever one wishes to spot check that a curious feature has actually been rejected.

Upon executing the task, one will see the following prompt for each order of each image. One may skip any orders at anytime.

```
Fit line 1 of data01.ec.imh w/ graph  (yes|no|skip|YES|NO|SKIP) (yes):
```

16.1. Pitfalls and Techniques

The major weakness of **continuum** is the inability to fit medium frequency features along the the continuum. Thus, the resulting normalized continuum may appear to have a wave to it with an amplitude of $3-5\%$.

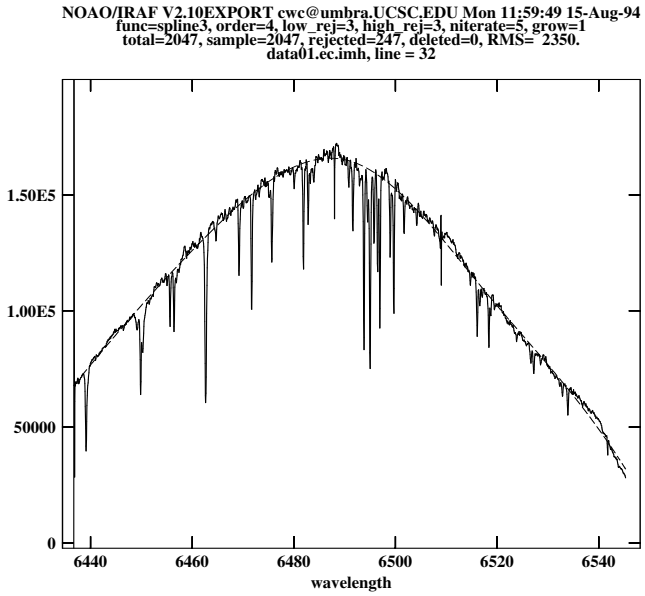


Figure 16–1. In the intensity verses wavelength window, one will notice that the blaze function may "weave" around the fitted function. When this happens, and it is difficult to remedy, the normalized continuum will be plagued by ripples (see Fig. 16–2).

One may attempt to remove this with higher order fits, but be very careful, as higher order splines may only aggravate the problem or modify line profiles. Also, it is best to keep "order" low on apertures that have lines with large wings. A good technique to avoid the influence of large winged features is to use the ":sample" parameter effectively (see Fig. 16–2) and concentrate on the continuum contiguous to the features of scientific interest.

The second major issue is that the normalization may be a consistent few percent high or few percent low from unity, depending upon one's fitting systematics. Be careful here. Inspecting the

fit in the "ratio" window ("k" key) and with the param "markrej" set to "no" allows one to avoid this problem, which is fine tuned using the ":high" and ":low" colon commands.

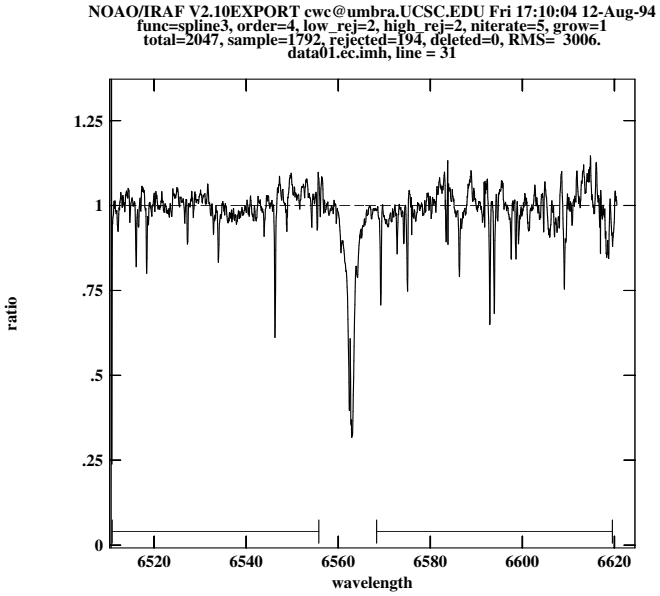


Figure 16–2. When viewing the ratio, the normalized continuum, one sees how poorly it may normalized. It may be difficult to obtain a gratifying normalization along the entire order using the **icfit** routines. Note the use of ":sample". The Hα line has been excluded from the fit.

APPENDIX A. IRAF V2.10.2 Bugs!

Below are a few pertinent "bugs" in the **apextract** package. These bugs are supposed to be exterminated in V2.10.3, but remain cautious of new ones!

A.1. APEXTRACT and the GAIN Parameter

When modeling aperture profiles, the optimal extraction routine will apply a Poisson noise model if one chooses variance weighting (enforced feature of **apflatten**). Variance weighting has been addressed multiple times in this manual. See Appendix B for a brief explanation of variance weighted extraction.

Since the noise model statistics are performed in units of e^-, the data in DN are multiplied by the gain factor, g. The cross dispersion profile model is actually a normalized probability distribution. But, V2.10.2 normalizes the profile in units of DN, not e^-. Consequently, the profile is improperly normalized, skewing the noise model. In the case of normalizing the flat field illumination functions, the resulting mean value turns out to be the gain instead of unity!

One may handle this in any number of ways, but here is a suggestion that has the benefit of a proper variance model and, in **apflatten**, proper normalization of the illumination profiles. Prior to performing the extraction (or illumination profile modeling), use **imarith** to multiply the image by the gain factor, effectively converting the data into e^-. Then, set the param (gain = 1) in the **apextract** task.

A.2. APNORMALIZE and the CENORM Parameter

The "cenorm" parameter in the **apnormalize** must be set to (cenorm = no). The output of **apnormalize** when (cenorm = yes) is entirely garbage. There is no work around for this bug.

A.3. CCDPROC and the CCDMEAN Card

This is not really a bug in **ccdproc**, but a bug in the bookkeeping of the **apflatten** task. Task **apflatten** normalizes one's flat field frame to unity, but does not update the CCDMEAN header card to reflect this new mean value. If one uses **ccdproc** to perform the flat fielding correction, then one needs to check that the CCDMEAN header card has value '1'.

Task **ccdproc** scales the flat field data by the value of the CCDMEAN header card. Be sure that this card's value accurately reflects the mean value of the flat field frame. If it does not, **hedit** the header of the calibration flat field frame prior to executing **ccdproc**. See §11 for further details.

APPENDIX B. Variance Weighted Extraction

Optimal extraction within IRAF is based upon the work of Marsh (1989) and Horne (1986).

The idea is to best determine the fraction of flux that falls into each cross dispersion pixel at every wavelength of the spectrum. This is achieved by optimally weighting the flux values in each cross dispersion pixel used in the extraction sum, minimizing statistical noise in order to maximize photometric accuracy.

Consider a two–dimensional image of an aperture ("spectrum") with cross dispersion direction pixels s and dispersion direction pixels λ. Given the following definitions, f_λ = the total flux at λ, $I_{s\lambda}$ = the signal counts, $B_{s\lambda}$ = the background counts, $W_{s\lambda}$ = the weighting factor, and $P_{s\lambda}$ = the probability cross dispersion profile normalized according to

$$P_\lambda = \sum_s P_{s\lambda} = 1$$

with all defined values in units of e^- ($= g \cdot \mathrm{DN}$), the f_λ can be determined. The $P_{s\lambda}$ give the probability that a photon with wavelength λ is registered in the cross–dispersion pixel s. For any given pixel s at each λ, f_λ is the scaled value

$$f_\lambda = \frac{I_{s\lambda} - B_{s\lambda}}{P_{s\lambda}} \equiv D_{s\lambda}$$

if no noise is present. When noise is present, the $D_{s\lambda}$ provide independent and un–biased estimates of the spectrum for every pixel with $P_{s\lambda} > 0$. The flux can be expressed as the unbiased linear combination of the $D_{s\lambda}$ in the cross dispersion direction:

$$f_\lambda = \frac{\sum_s W_{s\lambda} D_{s\lambda}}{\sum_s W_{s\lambda}}.$$

The goal of optimal extraction is to minimize the σ_{f_λ}. From basic error analysis theory, this is achieved when the $W_{s\lambda}$ are chosen to be

$$W_{s\lambda} = 1/\sigma^2_{D_{s\lambda}},$$

$$\sigma^2_{D_{s\lambda}} = \left(\sigma^2_{I_{s\lambda}} + \sigma^2_{B_{s\lambda}}\right)/P^2_{s\lambda} \equiv \frac{V_{s\lambda}}{P^2_{s\lambda}},$$

which follows from the definition of $D_{s\lambda}$. We have

$$W_{s\lambda} = \frac{P^2_{s\lambda}}{V_{s\lambda}}.$$

The optimal value of f_λ is then given by

$$f_\lambda = \frac{\sum_s P_{s\lambda}(I_{s\lambda} - B_{s\lambda})/V_{s\lambda}}{\sum_s P^2_{s\lambda}/V_{s\lambda}},$$

with

$$\sigma_{f_\lambda} = \left(\sum W_{s\lambda}\right)^{-1/2} = \left(\sum_s \frac{P_{s\lambda}^2}{V_{s\lambda}}\right)^{-1/2} .$$

Careful accounting based upon the actual reduction procedure to obtain the the $\sigma_{I_{s\lambda}}$ and the $\sigma_{B_{s\lambda}}$ would be ideal. However, the variance model used at this writing is a simple Poisson fluctuation and read–noise model. As stated, a proper noise model must be computed in units of electrons, e^-.

APPENDIX C. On the Hamilton Scattered Light

As has been addressed throughout this manual, the most nagging aspect of Hamilton reduction is a quantified measure of the intensity zero–point throughout the echelle format. A 2D two–component scattered light model for the Hamilton has been formulated and is in the process of being coded for release as a generalized IRAF scattered light removal task for echelle spectrographs.

This work is further detailed in Churchill and Allen (1995). The algorithm finds the "least–squares" scattered light surface by attempting to recover the interorder minima by modeling the background intensities as arising from both (1) a smooth global scattered light surface, and (2) a local wavelength dependent component from the nearest neighbor orders.

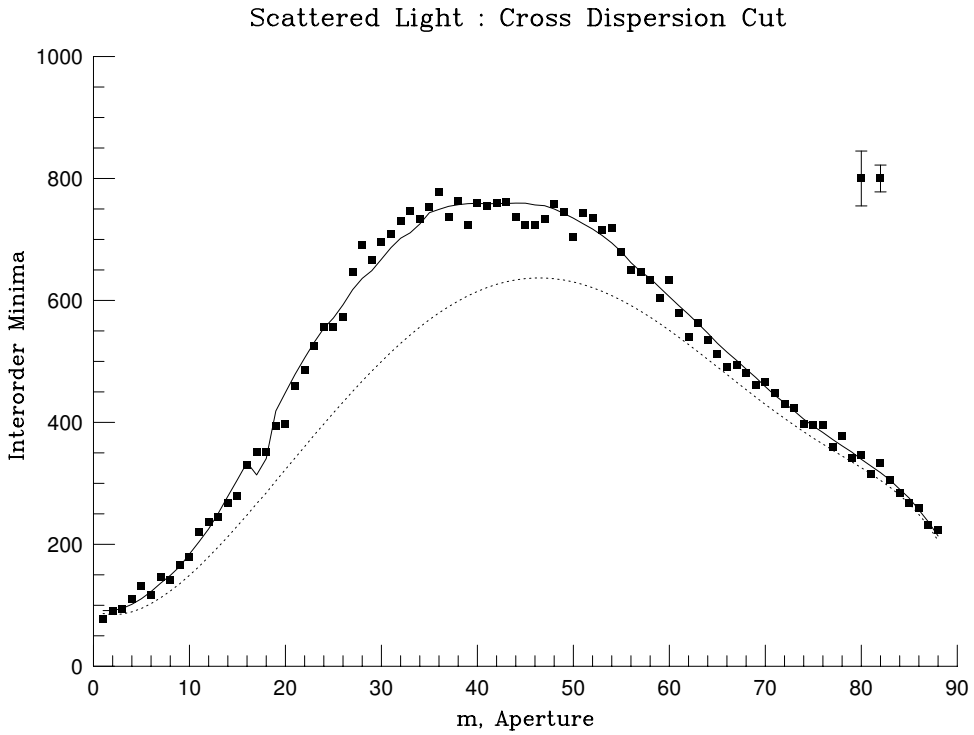


Figure C–1. The center–top Figure shows the interorder minima (dots) along column 81 of the water star verses the aperture number (order = aperture + 56). The dashed line is the "global" component, represented by an xorder=9 yorder=5 polynomial with cross–terms. The solid line gives the model interorder minima after the local component has been included. The two error–bars, from left to right, give the sigma of the data points at $m = 40$ and $m = 10$, respectively.

The resulting solution is a global two–dimensional surface with wavelength dependent local "ripples" (strong spectral features) from the contamination of the adjacent orders. The polynomial

orders are adjustable input parameters, denoted by n_x and n_y. The local component is based upon symmetric power–law tails for the cross dispersion illumination functions of the apertures.

This power law, denoted γ, is an adjustable input parameter to allow flexibility. Inverse square is a good canonical value. To treat possibly asymmetric profiles, the interorder minima of each order may be found using an "optimization" scheme, which determines a weighted mean fractional position between the adjacent order tracing solutions of the IRAF **apextract** database. This scheme is optional, the default interorder minima being the half–way point between the database tracings.

Presently, the model is a Fortran 77 module that links to a subsection of the SLATEC library (URL=http://www.netlib.org), and the IRAF IMFORT library (Tody 1986). IRAF images are operated upon, and an IRAF image of the scattered light surface is created.

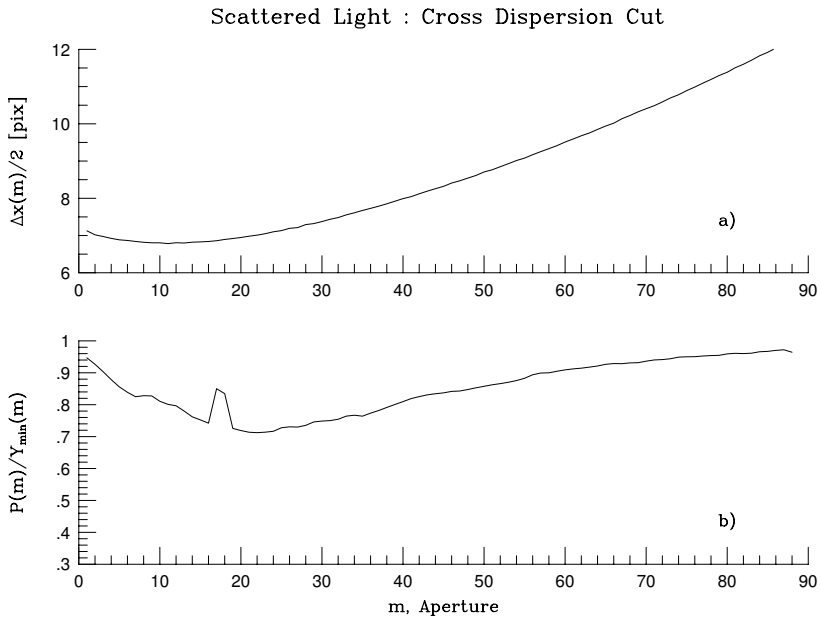


Figure C–2. Figure (a) shows the ratio of the "global" component to the model interorder minima. This Figure illustrates that the local (nearest neighbor) contamination is roughly 25% at Hα ($m = 31$), and that this contribution is simultaneously proportional to the nearest neighbor intensity and inversely proportional to the order separation. Column 81 was chosen because of strong telluric oxygen features at $m = 18$, which are apparent in the small peak in the "global" component. This illustrates that the local term is mitigated by neighboring absorption features. Figure (b) shows the half–order separation verses aperture number. The order separation on the Hamilton is a minimum at $m = 12$.

C.1. The Model

The work–horse code is a multi–dimensional non linear–least squares modified Levenberg – Marquardt algorithm. The code determines the n coefficients; $n-1$ for the two–dimensional polynomial and 1 for the local term. The coefficients are obtained by recovering the interorder minima intensities $F(x_{\min}, y)$ at each interorder location in both $x_{\min}$, the echelle format cross dispersion direction, and in y, the format dispersion direction. The least squares vector function to be minimized is

$$\begin{aligned} Y_i &\equiv F(x_{m,\min}, y) \\ &= P(x_{m,\min}, y) + C_n \left\{ f(x_{m,\max}, y) + f(x_{m+1,\max}, y) \right\} / \left(\Delta x_m / 2 \right)^{\gamma} , \end{aligned}$$

where the notation $f(x_{m,\max}, y) = F(x_{m,\max}, y) - P(x_{m,\max}, y)$ denotes the scaled maximum for order m, the $\Delta x_m = x_{m+1,\max} - x_{m,\max}$, and γ is the power law profile tail fall–off of the cross dispersion illumination function. The polynomial is a two–dimensional function with cross–terms, giving $n_x n_y = n - 1$ global component coefficients.

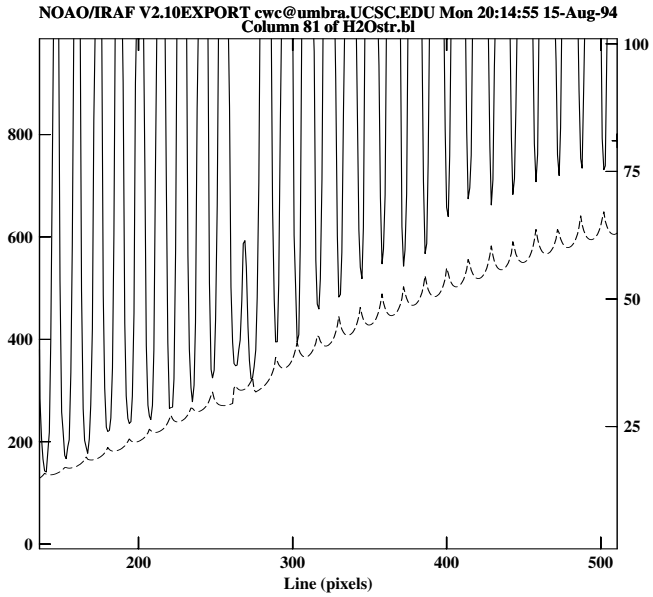


Figure C–3. Column 81 of the water star. The interorder minima (solid lines) of the cross dispersion profiles are shown for apertures 12 – 34, including the telluric oxygen feature at aperture 18. The resulting scattered light surface along this column (dashed line) is shown. Note the "global" trend of the surface and the power–law decay of the local "nearest neighbor" contamination. Also note that the model has clearly treated the strong absorption feature at aperture 18.

The order maxima positions $(x_{m,\max}, y)$ are taken from the IRAF **apextract** database, a file containing aperture models. The interorder minima positions $(x_{m,\min}, y)$ may be found using either (1) an optimization scheme, which is designed to account for possible profile asymmetries, or (2) simply assigning the mid–point of the **apextract** tracing solutions. The optimization scheme proceeds by determining the two lowest absolute minima positions between adjacent orders at each position along the dispersion, y. Denoted x^p and x^s, such that $F(x^p, y) \leq F(x^s, y)$, these minima are used to calculate a weighted mean fractional interorder position for each order m,

$$s_m = \frac{\sum_y w_y (x^p - x_{m,\max})/\Delta x_m}{\sum_y w_y},$$

where the $w_y = |2/(x^p_y - x^s_y)|$ are taken as the weighting factor. Then, the interorder locations are

$$(x_{m,\min}, y) = (x_{m,\max}, y) + (s_m, y).$$

For the Hamilton, the s_m are systematically less than the 0.5, the midpoint between order tracings. Note that the assumed power–law tails enforces symmetric profile shapes, but the outlined scheme provides for non-symmetric profile *domains*.

The model solution is computed with all data in units of e^- for proper statistical treatment. The σ vector, which provides the least squares weighting, is a variance model of the Y_i, incorporating Poisson fluctuations of the interorder minima, system read–noise, and uncertainty in the Δx_m. It is expressed

$$\sigma_i^2 = gF(x_{m,\min}, y) + RN^2,$$

where g is the amplifier gain in e^-/DN, RN is the read–noise in e^-.

C.2. The Background Surface

Once the n coefficients have been obtained, the background surface beneath each order m is constructed order by order, one dispersion pixel y at a time, from

$$B_m(x,y) = P(x,y) + C_n \left\{ \frac{f(x_{m-1,\max}, y)}{(x_{m-1,\max} - x)^\gamma} + \frac{f(x_{m+1,\max}, y)}{(x - x_{m+1,\max})^\gamma} \right\},$$

over the range $x_{m-1,\min} \leq x \leq x_{m,\min}$. At the interorder minima, fractional pixel contributions from the adjacent orders are computed. This "scattered" light surface is then subtracted from the data frame.

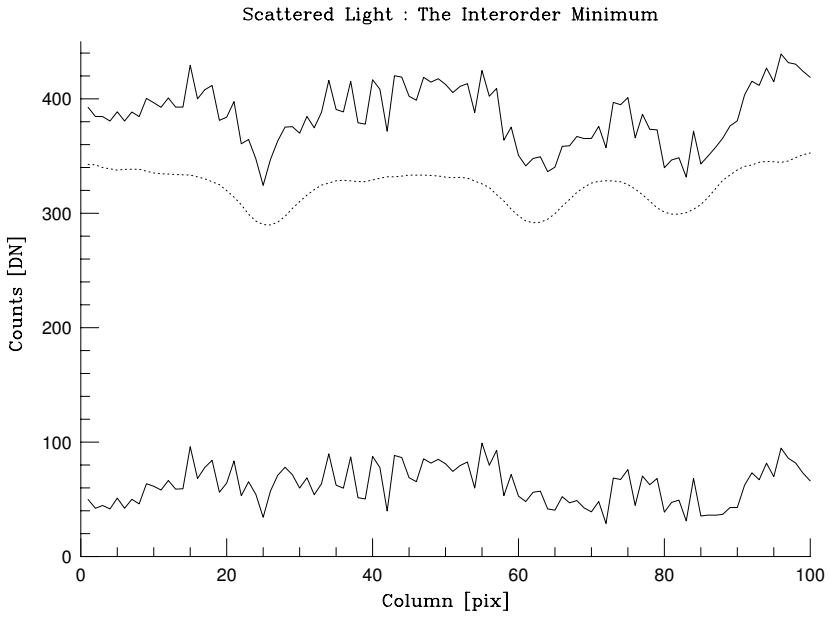


Figure C–4. The Interorder Flux verses Column of the water star. The solid line at 400 DN is the interorder spectrum (1 pix) of apertures 17 and 18. The dashed line is the scattered light model. The solid line at 50 DN is the resulting interorder spectrum following subtraction. Note that the contribution from the strong oxygen features has been fairly well removed.

The surface itself can be quite complex in the regions of strong absorption features, as shown in Fig. C–5.

C.3. Reduction Using this Model

If one used this model for scattered light removal (it is available from the author as a non–interactive Fortran code, Churchill 1994), the reduction steps would be effected as such:

(1) Use this model in place of the **apscatter** reduction step (§12)

(2) Perform the extraction as outlined in §13, but with the parameter (backgro = none) set.

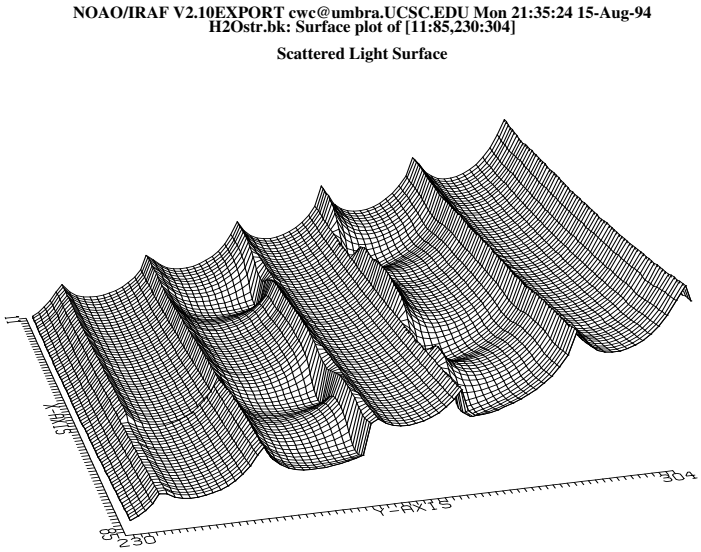


Figure C–5. Surface plot of the scattered light surface centered on the telluric oxygen features. The "peaks" correspond to the positions of the interorder regions, while the apertures correspond to the troughs. Note that the influence of the absorption features is such that less light is removed from the neighboring apertures and the amount removed obeys an inverse power law.

Though this model does not provide an *absolute* measure of the zero–point at each point in the echelle format, it does account for the nearest neighbor contamination quite effectively. The sensitivity to the choice of the power–law γ has been roughly estimated to be an 8% spread in the local contribution, where apertures are furthest apart (in the blue) and less where they are more tightly packed. This number was determined by varying γ from 1 to 3 and examining over–plots of Fig. C–2(b).

REFERENCES

Churchill, C.W. (1994), URL=http://ucowww.ucsc.edu/~cwc/hamilton/hamscatt/code.html

Churchill, C.W., and Allen, S.L. (1995), PASP, 107,193

Horne, K. (1986), PASP, 98, 609

Marsh, T. (1989), PASP, 100, 1032

Misch, T (1991), LOTR, 58

Shetrone, M. (1994), LOTR, 73

Tody, D. (1986), URL=ftp://iraf.tuc.noao.edu/iraf/docs/imfort.ps

Valdes, F. (1990), URL=ftp://iraf.tuc.noao.edu/iraf/docs/apex.ps

Vogt, S.S. (1987), PASP, 99, 1214