AUTOMATED IRAF REDUCTION SCRIPTS FOR ASTRONOMY GROUP AT BRIGHAM YOUNG UNIVERSITY

by

Craig A. Swenson

A senior thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Bachelor of Science

Department of Physics and Astronomy

Brigham Young University

April 2008

BRIGHAM YOUNG UNIVERSITY

DEPARTMENT APPROVAL

of a senior thesis submitted by

Craig A. Swenson

This thesis has been reviewed by the research advisor, research coordinator, and department chair and has been found to be satisfactory.

Date	Michael D. Joner, Advisor
Date	Eric G. Hintz, Research Coordinator
Date	Ross Spencer, Department Chair

ABSTRACT

AUTOMATED IRAF REDUCTION SCRIPTS FOR ASTRONOMY GROUP AT BRIGHAM YOUNG UNIVERSITY

Craig A. Swenson

Department of Physics and Astronomy

Senior Thesis

This thesis introduces a series of IRAF scripts written to facilitate the reduction of large amounts of data by students and faculty in the Astronomy Research Group of Brigham Young University. These automated reduction scripts provide a complete data-pipeline for frames taken using Brigham Young University's Orson Pratt and West Mountain Observatory telescopes. The functionality of these scripts is described in detail, along with the thought processes involved in their writing. Observations of the variable stars DY Peg and AE UMa are used as a test case to demonstrate the use of the scripts and to show improvements made in the reduction process. 17 new times of maximum light are presented for DY Peg.

ACKNOWLEDGMENTS

I would like to acknowledge all those who have helped me in my work on this thesis. In particular, I would like to thank my partner in this ambitious project, Paul Iverson. Without Paul I wouldn't have had the motivation to keep modifying and tweaking these scripts. The creation of SIDAP made both of us stretch ourselves beyond what either of us could do individually.

Also, I must thank Professor Michael D. Joner and Dr. Eric Hintz for their patience through the barrage of questions that accompany the development of a new reduction process, and though I've never met them, I need to thank Mike Fitzgerald and Frank Valdes for their insights into the inner workings of IRAF.

I would like to thank Jeremy Schoonmaker and Alie Porter for beta testing SIDAP and informing Paul and I of problems we were unaware of. I would also like to thank Lisa Joner for her time reading and editing this thesis.

Lastly, I want to acknowledge my wife Katie. Her love and support (along with not so subtle reminders) kept me working hard on completing this thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 IRAF Reduction Process

The basic steps involved in the processing of astronomical data are universal for the type of Charge-Coupled Device (CCD) cameras and telescope combinations currently owned and operated by Brigham Young University (BYU). These data reduction steps include bias-, dark-, and flat-field correction of the raw instrumental data. There are several programs available that will perform these reductions, each with unique advantages and disadvantages. Astronomy students and faculty of BYU use the Image Reduction and Analysis Facility (IRAF) for nearly all of their data reduction. IRAF is developed and maintained by the National Optical Astronomy Observatories (NOAO) in Tucson, Arizona, and is specifically written for use with astronomical data.

Each of the following reduction steps is necessary to remove unique noise characteristics of the CCD.

1.1.1 Bias Reduction

Bias frame reduction removes the inherent bias that may be displayed by individual pixels in the CCD. This bias is introduced as the pixels are read off the CCD in order to eliminate negative pixel values, but it needs to be removed later. The level of bias to be removed is determined by taking zero length exposures, which ensures that no photons, cosmic rays, or dark counts are registered by the pixel array. Several of these bias exposures are typically taken during each night of observations and are then averaged, night by night, to determine an average bias count per pixel. These averaged frames are then subtracted from the object frames to remove the imperfection.

1.1.2 Dark Reduction

Dark reduction is necessary in order to remove the thermal noise present in the CCD. The thermal noise, called "dark current", diminishes with temperature, but must be corrected for on the CCDs used by BYU, because their operating temperature is relatively high. The amount of dark current present, and the response to that current, varies from pixel to pixel. To remove this effect, a series of frames is taken without opening the shutter of the camera. This eliminates any actual photon counts from being recorded. The averaged bias correction is subtracted from these images, leaving only the counts recorded due to dark current. These corrected "dark frames" are averaged and then subtracted from the object frames.

1.1.3 Flat Reduction

A CCD operates by taking advantage of the photoelectric effect. Incoming photons strike a layer of doped silicon, which absorbs the photons and releases electrons. The linearity of a CCD refers to whether a single photon hit corresponds to the release of a single electron. CCD cameras are good tools for astronomical purposes because of their high linear response. Unfortunately, they are not absolutely linear. Correction for non-linear response can be achieved through the use of flat fields. Flat fields are frames taken by exposing the CCD to a uniform or "flat" light distribution and then measuring the response of each pixel. A perfectly linear CCD will have an equal number of electrons in each pixel. A non-linear CCD will have varying numbers of counts per pixel. To remove this effect, several "flat frames" are taken, and then corrected for both pixel bias and the dark current by subtracting the averaged bias frame and dark frame. The corrected flat frames are then averaged to produce a flat field correction frame. The object frames are divided by the flat field correction frame in order to scale their non-linear response.

1.2 Purposes for Writing Automated Scripts

BYU astronomy students are first introduced to IRAF in Physics 329. This class gives students the opportunity to learn how to use a telescope and CCD to obtain data, and how to use IRAF to process the data and obtain magnitudes. The process of learning how to use IRAF is one that takes patience and repetition. There are dozens of parameters that need to be set, and literally hundreds of possible different combinations of algorithms and parameters that could be used in order to produce frames of scientific quality from the raw instrumental frames. Beginning students are shown how to edit each necessary parameter from the *xgterm* command line, and are then required to enter each individual command to average bias frames, subtract the combined average bias frame from dark frames, average dark frames, subtract the combined average dark frame from flat frames, etc. This process of individually setting parameters and executing each command is essential knowledge for beginning students. Without this experience, students would never fully understand what IRAF does, how it works, and how to interpret their results. It is not uncommon for mistakes to be made, and it becomes nearly impossible to determine where the error was introduced without having an understanding of how each IRAF command works.

For more advanced students who know how to use IRAF, executing each of these commands individually becomes very repetitive and is not an effective use of research time. Particularly on very large data sets, a considerable amount of time is added to the reduction process if the student is required to enter each command individually. If a full night of telescope time is dedicated to observing a single object, there can easily be over 1000 frames to reduce. Handling the reduction process of such a large set, in addition to other obligations students have, can literally take days. The use of automated scripts, written in the IRAF cl language, can combine a series of IRAF tasks into a single command typed at the command prompt. By combining steps in this type of automated script, there is no lag time between the

running of individual IRAF commands, and large data sets can be completely reduced and magnitudes determined in the course of a few hours.

These types of automated scripts also allow the inclusion of UNIX commands. By combining familiar IRAF cl commands with UNIX commands, it is possible to produce scripts that work on data from a variety of CCD-telescope combinations. This is extremely useful at BYU, where we have large amounts of data from four different telescopes. Each telescope produces slightly different headers for their respective frames, meaning that parameters have to be changed inside IRAF when switching between telescopes. With automated scripts, the necessary parameter changes can be written into the script itself, and then automatically changed by the script as different situations arise.

Originally, these automated scripts were intended to condense the number of individual typed commands required to execute the basic IRAF reduction steps mentioned above. However, with each new script, it became an increasingly more trivial task to re-reduce data with a different set of parameters and algorithms. In a single day, researchers might experiment with multiple combinations of parameters and algorithms. Because of the decrease in time required, the scripts began taking on a new focus. They not only speed up the reduction process, but can also be used to determine the most effective method for reduction of a specific data set. New scripts were written and added to the reduction pipeline that were not part of the basic reduction process outlined above, but which provide better final results.

Chapter 2

Automated IRAF Scripts

2.1 General Description of Scripts

There are two methods of writing scripts within IRAF: procedure language and command language. Both methods offer distinct advantages as well as disadvantages. Procedure language allows access to all of the programming abilities offered by the native IRAF scripting language, such as error checking abilities, but UNIX commands are not recognized. Command language gives access to all of the regular IRAF tasks (i.e., *rfits, ccdproc, phot*, etc.), and also allows for the use of UNIX commands. For our purposes, the command language was the preferred option, because of the added power offered by the UNIX abilities.

The scripts can be broken up into two main categories: those that are observatory specific, and those that have been written to allow for use on any telescope-CCD combination currently being used by BYU. The observatory specific scripts are named to reflect either the observatory to which they apply, if the observatory has multiple telescopes, or a specific telescope, at observatories with only a single telescope. For example, scripts written for data obtained by telescopes at the West Mountain Observatory (WMO) are given the prefix 'wmo_', and will work with any of the three telescopes located at WMO. Data obtained on the David Derrick telescope (DDT) of the Orson Pratt Observatory are given the prefix 'ddt_', because there is only one telescope located at the observatory. Data from the Tenagra II telescope are given the prefix 'ten_' for the same reason. The remaining scripts, those that are not observatory specific, are given the prefix 'ap_' signifying that they are "all-purpose" scripts.

It may be difficult for a user to remember at what point they stopped during a reduction process when large data sets are involved, or when working with multiple

data sets at once. To help with this problem, some of the scripts rename the object files as they run. For example, after running the *ap_proc* script the object files are renamed with a '.proc.fits' extension, signifying that they are processed. The renaming of filenames has been reserved to those scripts that make permanent changes to the images. We felt it important, however, to have a method of knowing exactly what scripts have been run, including those which don't rename the files. To achieve this, every script adds a new keyword to the header containing the date and time the script was run. The keyword added to the header is the name of the script that was run (i.e., *ap_proc* adds the header keyword AP_PROC).

In order to make the running of these scripts as simple as possible, care has been taken to set parameters "on the fly" within the script itself, thus eliminating the need to *epar* multiple IRAF tasks before reducing data. It is still recommended that users task and execute Dr. Eric Hintz's *newton* script, because a few of the parameters defined in *newton* are still used in the new reduction methods. Those parameters that carry over from *newton* are not necessarily set as part of any of these scripts.

The current script versions (at time of this publication) use what seem to be the best methods available for data reduction. As further investigations are made into data reduction methods, it may become necessary to update the automated scripts to reflect new findings.

2.2 Detailed Description of Scripts

This section provides a detailed description of each of the scripts. The descriptions include lists detailing which IRAF commands, normally entered individually at the command line, are covered by using the specific script. Also discussed are the reasons for using the chosen parameters and algorithms. These scripts are not intended to be run separate of each other, but rather consecutively as an entire data reduction package. The Swenson-Iverson Data Analysis Package (SIDAP) consists of 12 scripts, or groups of scripts, which are presented here in the order that they must

be executed. Three additional scripts are also presented, after the 12 SIDAP scripts, which are not necessary but useful nonetheless. The name given for each script is the root name, leaving off any associated version number in anticipation of further versions. All scripts are presented in their entirety in Appendix A.

2.2.1 *ap_rfits*

This script replaces the regular IRAF *rfits* command. The purpose for this script is to create a uniform naming convention that will be used throughout the remainder of the scripts. Bias frames are named 'zero-', followed by the file number. Dark frames are similarly named 'dark-', followed by the file number. Flat frames are named 'flat(filterletter)-', followed by the file number (i.e., flatV-1.fits). Object frames are named using a user defined name preceded by 'obj-' (signifying that it is an 'object' frame), followed by '-' and the file number (i.e., obj-dypeg-1.fits). In order to run *ap_rfits*, the script makes the assumption that all of the object frames for a specific object and all necessary calibration frames are in the same folder. Because *ap_rfits* was written to work specifically with large data sets, it only accepts a single user input for the name of all object frames. The script moves the raw frames into a folder named *filesrawfit*, which is a subdirectory of the current folder.

2.2.2 *ap_calhead*

The *ap_calhead* script fixes the headers for all of the calibration frames. This includes changing the IMAGETYP in the image headers to either "zero", "dark", or "flat". It also adds SUBSET to the flat frame headers and sets it to the appropriate filter letter. All of these changes are achieved using the IRAF *hedit* commands.

2.2.3 *(∗)_obhead*

There are three versions of the *(∗)_obhead* script that are responsible for editing the headers of object frames. Each of the three are observatory specific (as outlined in Section 2.1), but perform essentially the same function. The *wmo_obhead*

script works for all three telescopes at WMO, with the user being prompted to enter which specific telescope was used to obtain the frames. Both the *ddt_obhead* and the *ten_obhead* run without user input. The *(∗)_obhead* scripts replace the use of a '.cmds' file and the *asthedit* command. If the right ascension and declination are already in the header, the script accepts these values as being correct. If neither of these values exist in the header, the user is prompted to enter the right ascension and declination, which the script places into the header under RA and DEC. The script also sets SUBSET to the filter used, EPOCH to "2000", OBSERVAT to the appropriate observatory, OBSERVER to "BYU", and IMAGETYP to "object". The IRAF commands *setairmass* and *setjd* are also executed by the *(∗)_obhead* scripts.

In order to calculate the correct airmass and Julian date, the header must include correct values for the universal time and the sidereal time. There are two different formats used to report the universal time in the image header, depending on what software is used. In order to make the scripts compatible with both formats, the *imgets* command is used to extract the universal time from either the DATE-OBS or TIME-OBS image header keywords (whichever keyword exists for the particular frame), and a new header entry called UT is created with the universal time. Although this process may seem redundant because the universal time is already contained in either DATE-OBS or TIME-OBS, having a separate UT header keyword is necessary to simplify the sidereal time calculation, and the *setairmass* and *setjd* tasks.

The *wmo_obhead* was also specifically designed to handle data taken at WMO during the summer and fall of 2006. During this period of time, image headers reported the local time rather than the universal time. After the script extracts the universal time and calculates the sidereal time, it displays the universal time given in the header, the calculated sidereal time, and the right ascension of the object. The user is prompted to enter whether the given universal time is correct or whether it is the local time. If the universal time is correct, the script continues. If the given universal time is incorrect, the user is prompted to input the correct hour difference

(dependent on daylight savings time) and the script modifies the UT header value appropriately, recalculates the sidereal time, and then continues.

2.2.4 *ap_gain_rdnoise_(∗)*

There are three separate scripts involved with the calculation of the CCD gain and readnoise. It is possible, and often convenient, to execute these three scripts using a single script (i.e., some sort of *ap_gain_rdnoise_master*), but the scripts were written independently of each other and are presented here in their independent form. The three scripts will likely be officially combined at a future time to ensure uniformity of data reduction among students.

The calculation of the gain and readnoise has not historically been a part of the basic IRAF reduction procedure used by BYU students. However, these calculations are crucial in order to obtain the best possible reduction, and are mandatory if using SIDAP.

i. *ap_gain_rdnoise_collector*

The purpose of the *ap_gain_rdnoise_collector* is to take the raw bias and raw flat frames, then calculate the gain and readnoise. The script determines the total number of flat frames available and prompts the user to input the number of the available frames to be used. The same is done for the bias frames. Rather than using the *findgain* task included with the IRAF distribution, users of SIDAP should task a special *ap_findgain* command in their 'login.cl'. This separate *ap_findgain* was created under the advice of Frank Valdez (Valdez 2007) because of problems arising in the verbose features of the IRAF *findgain*. The only difference in the *ap_findgain* is the exclusion of the verbose output.

The gain and readnoise values are calculated for five different sections of the CCD. Each section is a square 100 pixels by 100 pixels in size. The sections are placed across the CCD chip with one in each of the four corners of the CCD and the final section centered in the middle of the chip. This distribution gives a

large enough sampling of the entire chip to give accurate results, and speeds up the time required to calculate the gain and readnoise. The gain and readnoise values from each of these calculations are dumped into a gainrdnoise.info file.

ii. *ap_gain_rdnoise_calculator*

This script takes the gainrdnoise.info file created by *ap_gain_rdnoise_collector*, averages all the gain values and readnoise values, and produces a gain.info and rdnoise.info file. These two '.info' files contain the single averaged value for the gain and readnoise, respectively, as well as the standard deviation of those averages.

iii. *ap_gain_rdnoise_obhead*

In order to run *ap_apphot*, the gain and readnoise are required to be in the object frame headers. Even for those who do not wish to use *ap_apphot*, the gain and readnoise are also used in the IRAF *phot* package inside of *datapars*. Historically, these values have been left blank. To use the *ccdclip* rejection algorithm inside *zerocombine*, *darkcombine*, and *flatcombine*, the gain and readnoise need to be present in the bias, dark, and flat field headers as well. (See discussion on *ccdclip* and other rejection algorithms under *ap_proc*.) Not knowing what rejection algorithm is going to be used, *ap_gain_rdnoise_obhead* uses *hedit* to add the gain and readnoise to the headers of the object frames, as well as the calibration frames under the GAIN and RDNOISE header keywords, respectively.

2.2.5 *ap_pierside*

All of the telescopes currently used by BYU must flip over the pier when observing an object on both sides of the meridian. The exception to this is the DDT, which can observe objects near the celestial equator continuously without flipping. When a telescope goes over the pier, it rotates the CCD by 180°. Historically, this has meant that two different coordinate files must be created, one for each side of

the pier. Use of the IRAF *rotate* command can correct for this 180° rotation, and make the frames appear continuous. The *ap_pierside* script is designed to separate the images into the two different orientations to simplify their rotation.

This script requires the most user interaction and is the least automated of the group. The script determines, based on exposure length, a window of time during which the telescope needed to be moved over the pier. Any exposures prior to this window are defined as 'east' and those after it are defined as 'west', with 'east' and 'west' referring to which side of the meridian the object was located. Pier flips do not always occur right at the meridian, so the first few frames west of the meridian may actually be in the 'east' orientation. For frames taken during the window of time when the telescope was flipped, each frame is displayed in ds9 and the user is prompted as to whether the frame in question is an 'east' oriented frame or a 'west' oriented frame. As soon as the user declares a frame to be in the 'west' orientation, the script assumes that the remainder of the frames taken will also be 'west' oriented.

The script makes two modifications, one to the header and the other to the file name, to distinguish 'east' frames from 'west' frames. A new header keyword, PIER, is added with either the letter 'E' or 'W', depending on the orientation. The file names are also modified to include '-east' or '-west' immediately before the '.fits' extension.

2.2.6 *ap_proc*

One of the most crucial changes to the basic IRAF reduction process occurs in *ap_proc*. The IRAF commands *zerocombine*, *darkcombine*, *flatcombine*, *mkillumcor*, and *ccdproc* are all run as part of *ap_proc*. Within *zerocombine*, *darkcombine*, and *flatcombine*, the user must specify the desired rejection algorithm. This rejection algorithm determines what pixels should be thrown out before making a combined averaged frame. There are seven different rejection algorithms to choose from: *none*, *minmax*, *ccdclip*, *crreject*, *sigclip*, *avsigclip*, and *pclip*. By default, IRAF uses the

minmax rejection algorithm. BYU students have generally used either this algorithm or *avsigclip*.

The user is also prompted as to what type of "Flat algorithm" they would like to use in their reduction. The two options given are "Calibration" or "Object". Calibration refers to the normal flat reduction using flat field frames, whereas Object refers to using a sampling of the object frames to create an artificial flat frame which can be used for flat field reduction in the absence of flat fields.

Regardless of which "Flat algorithm" is chosen, an illumination frame is also created and applied to the object frames. Illumination frames are created by combining multiple flat frames while drastically smoothing out the pixel to pixel variations in order to see the large scale CCD illumination (*mkillumcor* help file). The addition of illumination field correction is a major change to the processing and has not historically been used at BYU. The addition of illumination correction is more for thoroughness than for significant improvement of the images. The effect seems to be negligible on the current BYU telescopes, but it may be something that will be of use in the future. Illumination corrections were added at the suggestion of Professor Mike Joner (Joner 2008).

In addition to processing all of the frames, the script also copies the unprocessed frames to a new folder called *filespreproc*, which is a subdirectory of the current folder. With the unprocessed frames easily available, users can reprocess the frames using a different algorithm or different parameters, without having to completely start the script sequence over with *ap_rfits*. The script also changes the file names of the processed frames to make them easily distinguishable, giving them a '.proc.fits' suffix instead of just '.fits'.

As stated, BYU students have generally used either *minmax* or *avsigclip* for the rejection algorithm. Both of these rejection algorithms are useful, in certain circumstances, but they may not be the most rigorous choices. The IRAF help file for the *combine* task states that the *ccdclip* algorithm is "the best clipping algorithm to use if the CCD noise parameters are adequately known". We recommend the use

of the *ccdclip* algorithm, but because there may be times where it is necessary to use other algorithms, the script gives the user the option of choosing between *minmax*, *avsigclip*, and *ccdclip*. The other four algorithm options are used so infrequently that they are not currently an option provided by SIDAP.

A brief description of the three algorithms available in SIDAP is given to help users understand why the *ccdclip* algorithm is recommended. (To see numerical comparisons between reduction methods, see Section 3.1 in the next chapter).

i. *minmax*

The *minmax* algorithm simply rejects a user specified percentage of the high and low pixels, determined by the *nlow* and *nhigh* parameters. If the CCD varies greatly from frame to frame, then the quality of the combined frames would depend highly on the values of *nlow* and *nhigh*. For example, if *nhigh* is set to reject a low percentage of pixels, it may not reject a large enough sample and the frames would not be adequately reduced. On the reverse side, if the CCD being used is exceptionally consistent in its bias level, then *minmax* will unnecessarily discard pixels.

ii. *avsigclip*

The other commonly used algorithm, *avsigclip*, is designed to compensate for both of the aforementioned problems that can arise by using *minmax*. The algorithm rejects pixels based on a comparison to an estimated σ (standard deviation) of each pixel. This σ is calculated using Eq. 1.

$$\sigma_{(column,line)} = \sqrt{\mathrm{GAIN}_{(line)} \times \mathrm{SIGNAL}_{(line,column)}} \tag{1}$$

The $\mathrm{SIGNAL}_{(line,column)}$ is estimated using either the mean or median value (user defined parameter) of each (*column,line*) (i.e., pixel). The $\mathrm{GAIN}_{(line)}$ is estimated for each line along the read out axis of the CCD by scaling the square of the residuals along the line with the mean/median. The scaling factor is the

estimated $\mathrm{GAIN}_{(line)}$. Theoretically, the CCD gain is a constant number across the entire chip. However, the $\mathrm{GAIN}_{(line)}$ may vary due to the fact that it is simply an estimate.

The parameters *lsigma* and *hsigma* define the range of acceptable pixel values. Any pixel value higher than $hsigma \times \sigma_{(column,line)}$ or lower than $lsigma \times \sigma_{(column,line)}$ is rejected. After rejecting pixels, the $\mathrm{SIGNAL}_{(line,column)}$ is recomputed with the new mean/median and the process is repeated. The $\mathrm{GAIN}_{(line)}$ is not recomputed during each successive iteration. This process is repeated until no more pixels are removed, or the number of total pixels remaining falls below the user defined parameter *nkeep*. Currently, *lsigma* and *hsigma* are set so as to retain about 55% of the pixels (assuming a Gaussian distribution), and *nkeep* is set to half the number of input images.

The *avsigclip* algorithm has some obvious advantages over the *minmax* algorithm. By using *avsigclip*, the rejection algorithm isn't limited to rejecting only a set percentage of the pixels if it determines that a larger number should be rejected. Also improving on the deficiencies of *minmax*, *avsigclip* is not required to reject any pixels if they all fall within σ. The iterative nature of the algorithm also gives it an advantage, as it is able to weed out bad pixels and obtain a more accurate solution.

Limitations still exist with this method, however. The estimation of $\mathrm{GAIN}_{(line)}$ can be easily skewed by a handful of erroneous pixels. Since the $\mathrm{GAIN}_{(line)}$ is not recalculated after the first iteration, any errors introduced at the beginning will propagate through the data. Another limitation is that this algorithm doesn't take into account the readnoise of the chip, which will also adversely affect the estimated $\mathrm{GAIN}_{(line)}$.

iii. *ccdclip*

Just as *avsigclip* is able to compensate for the limitations of *minmax*, the *ccdclip* algorithm is able to improve on the weaknesses of *avsigclip*. The *ccdclip*

algorithm has generally not been used by BYU students because it requires that a good value for both the CCD gain and readnoise be known. In the past, students have thought of the calculation of the gain and readnoise as something that was difficult to do (an unfortunate misconception that somehow was propagated down through generations of students). Because of this misconception, using the *ccdclip* algorithm hasn't been a technique utilized until now. The previously discussed set of *ap_gain_rdnoise_** scripts makes the calculation of the gain and readnoise a trivial exercise that requires nothing more than a little time.

The main difference between *ccdclip* and *avsigclip* is the method by which σ is evaluated. The equation used by *ccdclip* to calculate the standard deviation is given in Eq. 2.

$$\sigma = \sqrt{(\text{RDNOISE} \div \text{GAIN})^2 + \text{SIGNAL} \div \text{GAIN} + (\text{SNOISE} \times \text{SIGNAL})^2} \tag{2}$$

The RDNOISE and GAIN values are taken from the header values. The SIGNAL is determined for each pixel over the set of frames being processed, just as in *avsigclip*. The SNOISE is the sensitivity noise value of the CCD. If this value is unknown (as it currently is for the CCDs being used by BYU), then the value can be set to 0 (*combine* help file). This determination of σ is superior to that of *avsigclip* because it uses the actual CCD gain instead of estimating the value during the rejection procedure. It also takes into account the readnoise, something that *avsigclip* ignores completely.

Once σ is calculated, pixels are rejected if their value is higher than $hsigma \times \sigma$ or lower than $lsigma \times \sigma$. After eliminating outliers, *sigma* is recalculated without the rejected pixels and the process is reiterated. Reiteration occurs until no remaining pixels are rejected. Should the number of pixels remaining

fall below the value of the parameter *nkeep*, the iterations stop and the last *nkeep* number of pixels are used. Currently, *lsigma* and *hsigma* are set so as to keep about 55% of the pixels (assuming a Gaussian distribution), and *nkeep* is set to half the number of input images.

2.2.7 *ap_skynoise*

The *ap_skynoise* script determines an accurate value of the sky background and the deviation of the sky for each frame. The determination of the sky background is mostly for analysis purposes (i.e., seeing how the sky brightness fluctuates over the night). The deviation in the sky background, however, is a parameter in *datapars*, called *sigma*, that needs to be set in order to run *phot* with greater precision. Historically, the default value, INDEF, has been kept. The parameter *sigma* is not only used by the centering algorithm in *phot*, but also by the finding algorithm inside *daofind*.

In order to calculate precise values for the sky background and the sky deviation, a relatively large sample size is needed, distributed across the CCD. The IRAF task *daofind* is used when *skynoise* is run, using estimated values for *sigma* and the *threshold* parameter in order to find candidate stars to use in calculating the sky background. If less than 25 candidates are found, then the value of *sigma* is lowered and *daofind* is run again. If more than 100 are found, then the value of *sigma* is raised and *daofind* is run again. The reason for limiting the number of candidate stars is to limit the total script execution time. Once a suitable number of candidate stars have been found, *phot* is run using a set *annulus* of 10 pixels, and *dannulus* of 4 pixels. The sky background and sky deviation values are extracted from the '$*$.mag.1' files for all of the object frames. These values are averaged and the average sky background and average sky deviation are inserted into the header under SKYAVG and SKYDEV, respectively.

Some may worry about running this script on a frame with fewer than 25 stars, because of the required minimum candidate number. Such concern is unnecessary,

because eventually *sigma* will be lowered enough that hot pixels will be interpreted as stars and the sky will be calculated in a ring around the hot pixel.

2.2.8 *ap_fwhm_obhead*

Originally, *ap_apphot* was written with the intent of using an aperture equal to the full-width at half-maximum (FWHM), with the FWHM calculated for each frame using a weighted average, as calculated by *psmeasure*. As *ap_apphot* was developed further, it moved away from this method. Nonetheless, *ap_fwhm* script was retained for analysis purposes and to define another *datapars* parameter. The parameter *fwhmpsf* is used in *daofind* to eliminate non-stellar objects from its finding routine. This increases the precision of *daofind*, which then increases the accuracy and speed of the *ap_align* script. The *fwhmpsf* is also used by the *ofilter* and *gauss* centering algorithms. The weighted average FWHM of each frame is written to the image header under the keyword FWHM.

2.2.9 *ap_rotate*

With the object frames split into 'east' and 'west', the *ap_rotate* script can be used to rotate either the 'east' or 'west' frames so that all the frames are in the same orientation. The user is allowed to choose which set of frames will be rotated by supplying the script a character string specific to that set. This design allows for the script to be used in situations other than those where the frames are separated into 'east' and 'west'.

2.2.10 *ap_trim*

The *ap_trim* script is designed to remove a set number of pixels around the entire edge of the frame. The default value is set to 10 pixels. This trimming of the frames is required to eliminate possible overscan regions along the edges of some CCDs. These overscan regions (regions of no data but still containing biased pixels) are interpreted by *daofind* as high concentrations of stars. These misinterpreted

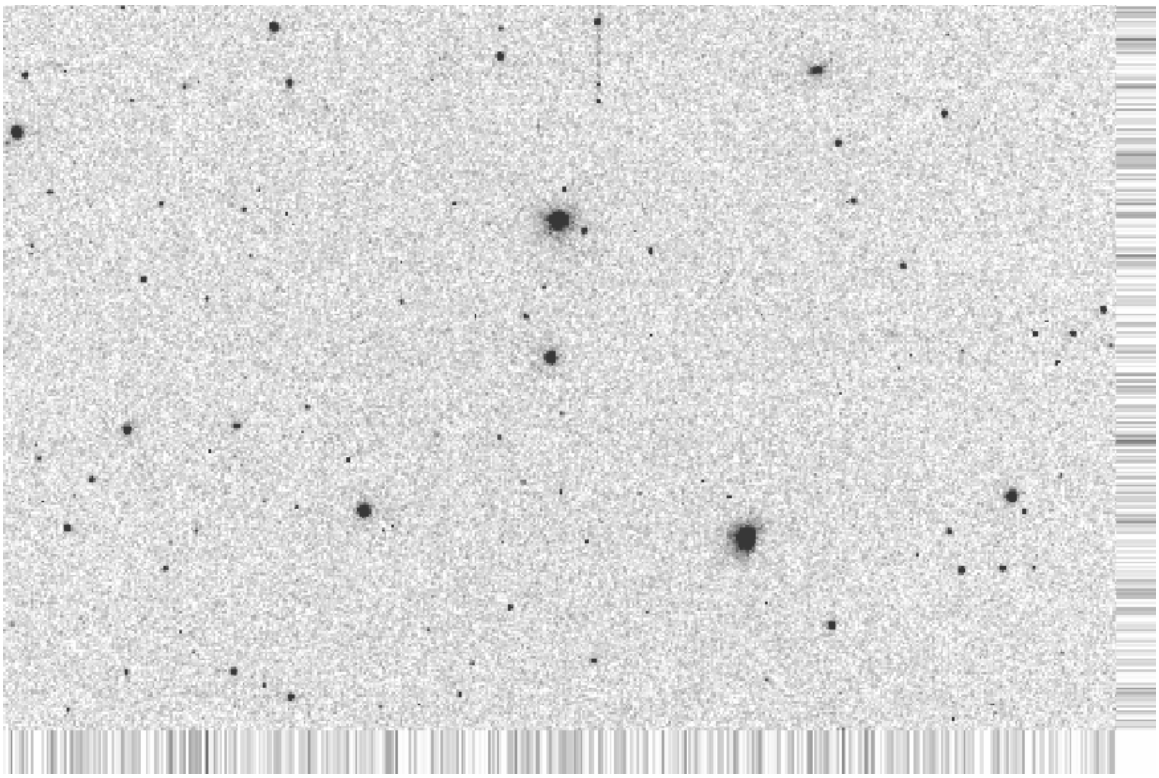

Figure 2.1: Example of an aligned frame

points result in "floating point errors" when running the *ap_align* script, making it impossible to run *ap_align* with the overscan region included.

2.2.11 *ap_align*

This is the only script that was not programmed primarily at BYU. The *ap_align* script actually calls another script *autoalign* (Ofek 2008). The *autoalign* script takes advantage of two other scripts also written by Ofek, *autodaofind* and *xyshift.f*, which use *daofind* to create a coordinate file for each frame and then calculate the shift between coordinate files from frame to frame. The frames are then reoriented according to the calculated shift, relative to the first frame. This reorientation results in the pixels along the edge of the frame being dragged to fill in the empty space created. See Figure 2.1 for an example of an aligned frame.

The amount of shifting required, in both the x and y directions, is written into the header under X_PIXEL and Y_PIXEL. These X_PIXEL and Y_PIXEL values are then used by *ap_apphot* in determining whether a given star has drifted off the frame. The newly aligned frames are given an 'a-' prefix, to signify that they are aligned frames. The unaligned frames are copied to a new folder, *filescomp*, which is a subdirectory of the current folder.

2.2.12 *ap_apphot*

A very basic description of *ap_apphot* will be given here. For a more detailed description, including comparisons with other photometric methods, please see Senior Thesis written by Paul Iverson (Iverson 2008).

The *ap_apphot* script offers two different methods of determining the photometric magnitude of stellar objects. One method uses an integrated flux technique, and the other uses a multiple of the stellar FWHM. Both methods take advantage of differential apertures on a per star basis. By using differential apertures, the script ensures that equal percentages of each star are included in the magnitude determination, regardless of seeing conditions. Historically, BYU students have simply used large apertures, regardless of brightness, in order to compensate for varying seeing conditions. This results in an unacceptably low signal-to-noise ratio (S/N) for faint stars. Ideally, smaller apertures would be used to increase the S/N for these faint objects, while retaining the larger apertures for brighter sources. By using varying aperture sizes on stars of different magnitudes, it would be necessary to apply aperture corrections to bring all the stars to an equal level (Da Costa 1992). This is something that isn't done at BYU. By using *ap_apphot*, smaller apertures can be used, yielding a higher S/N, without having to employ aperture correction techniques. The use of differential aperture sizes replaces the need for aperture corrections. The script uses the IRAF *psfmeasure* and *phot* commands in determining the appropriate aperture size and the stellar magnitude.

The way in which the sky value around the star is determined has also been modified. Traditionally, set values were used for both the *annulus* and *dannulus* in *datapars*, with the *annulus* being at least as large as the actual aperture and the *dannulus* set to a value of 4 - 6 pixels. The use of a large *annulus* and *dannulus* limits how close two stars can be on the frame without an overlap of their annuli. If seeing conditions are good enough, an observer should be able to use a smaller *annulus* and *dannulus* and achieve magnitudes for stars that are close to one another. When *ap_apphot* was first written, the *annulus* was set using a multiple of the weighted average FWHM, as calculated by *ap_fwhm_obhead* with a constant *dannulus*. This was done to take advantage of good seeing conditions. It was pointed out, however, that by using this method, frames with good seeing – and consequently a small FWHM – are effectively penalized in their sky calculation by severely limiting the number of pixels included by the *dannulus* (Joner 2008). Nights of poor seeing (large FWHM) would in turn have a much larger number of pixels included in their sky calculation. In order to place all observation conditions on equal footing and still take advantage of good seeing conditions, it was decided to always include approximately 500 pixels in the sky calculation. This is achieved by having an *annulus* set to a multiple of the FWHM and varying the *dannulus* in incremental steps. This is done by reading out the number of pixels being used from the magnitude file, adjusting the *dannulus* size accordingly, and running *phot* again.

When *ap_apphot* is run, the user is prompted for a specific character string unique to the object frames. Typically, "a-∗" is used because of the way that *ap_align* renames frames. The user is also prompted for the name of the coordinate file (i.e., ds9.reg), the photometry method to be used, and whether the frames should be separated by pier side. The inclusion of the option to separate frames based on pier side is for frames that are taken using a CCD with a gradient. Both the WMO .31m and .4m CCDs have known gradients. Without separating the two pier sides, the errors calculated in a program such as *varstar* will be incorrect because of the shift introduced across the pier break due to the gradient.

It is assumed that the user will be using the *varstar* program, written by Dr. Eric Hintz, to produce differential magnitudes. As such, *ap_apphot* outputs '.lst' files and automatically removes any INDEF values, replacing them with the IRAF *zmag* value. The '.lst' files are named to uniquely distinguish them by filter, exposure time, pier side, and photometry method. For example, 20-second exposures on the east side of the pier, taken in the V filter that used the integrated flux method to include 95% of the flux, would receive the name "starV_20._E_flux95.lst".

It is important to note that the *ap_apphot* script requires that all of the previously described scripts will have already been executed. Otherwise, pertinent header information will not be available and the script will crash.

2.2.13 *ap_reg_apphot*

The *ap_reg_apphot* script was written simply as a way to quickly produce stellar magnitudes using the set aperture method that has traditionally been used at BYU. These magnitudes have been used for comparison purposes only, to judge the benefits obtained by using the newly developed integrated flux method of photometry.

The script prompts the user for a character string unique to the object frames, the name of the coordinate file, the size of aperture to be used, and whether to separate the frames by pier side. The script produces '.lst' files, just as *ap_apphot* does, replacing all INDEF values with the IRAF *zmag* value, and naming them in an identical manner as *ap_apphot*.

2.2.14 *ap_imagecheck*

The *ap_imagecheck* provides a simple way of quickly viewing large numbers of frames, determining if any need to be removed, and providing a way to permanently delete those images. When the script is run, the user is prompted for a character string unique to the set of images being examined. The images can be viewed either individually or in sets of 16 at a time. In both instances, the SAOImage DS9 program is used to view the images. If the single frame method is chosen, then following the

display of each frame the user is given the option to continue to the next frame, delete the current frame, or quit the script. The multiple frame method is similar. After each set is displayed, the user is given the option of deleting any of the current frames, continuing on to the next set, or quitting the program.

Although this script is not required to be used during the reduction procedure, it is available for the convenience of individuals working with large numbers of images in IRAF.

2.2.15 *ap_subtrim*

The *ap_subtrim* script is a user interactive version of the *ap_trim* script. The user specifies lower and higher x and y dimensions, and the script crops the object images at the specified dimensions. The trimmed files are output with a '.sub' file extension so as not to permanently alter the original images. This script can be useful when running scripts such as *ap_fwhm_obhead*. If an extremely bright, over-exposed star dominates the frame, the calculated FWHM will be significantly influenced by that star (*psfmeasure* uses a weighted average when calculating the FWHM). A more accurate representation of the FWHM can be determined if the frame is trimmed so as to remove the bright star, and the trimmed '.sub' files are used to determine the FWHM.

Although this script is not required to be used during the reduction procedure, it is available for the convenience of individuals working with images that need to be trimmed to a specific size.

Chapter 3

Application of Scripts to Data

3.1 Comparison of Reduction Methods on WMO .31m and DDT Data

The IRAF help file for the *combine* task states that *ccdclip* is the best reduction algorithm to use if the CCD has well-defined noise parameters. These parameters include the gain and the readnoise. We feel that the *ap_gain_rdnoise_(∗)* scripts give an effective means for establishing acceptable values of the gain and readnoise for all of the CCDs operated by BYU. This means that we should be able to take advantage of the benefits of using *ccdclip*. We conducted several different tests in an attempt to show how one reduction algorithm compares to another, and to confirm that *ccdclip* is the best option.

The first tests involved reducing the same night of data three times, using the *minmax*, *avsigclip*, and *ccdclip* reduction algorithms once each. All other reduction methods and parameters were held constant. Magnitudes were determined using the *phot* task, and *varstar5* was used to produce differential magnitudes. The hope was that an inferior reduction algorithm would produce results that contained more noise. The results were not what we expected. Table 3.1 shows the standard deviations of

Table 3.1. Standard Deviation of Differential Magnitudes for Star 1 in the Field of DYPeg Using Different Reduction Algorithms (WMO .31m)

Reduction Algorithm		σ
minmax	···	0.009372
avsiglcip	···	0.009516
ccdclip	···	0.009330

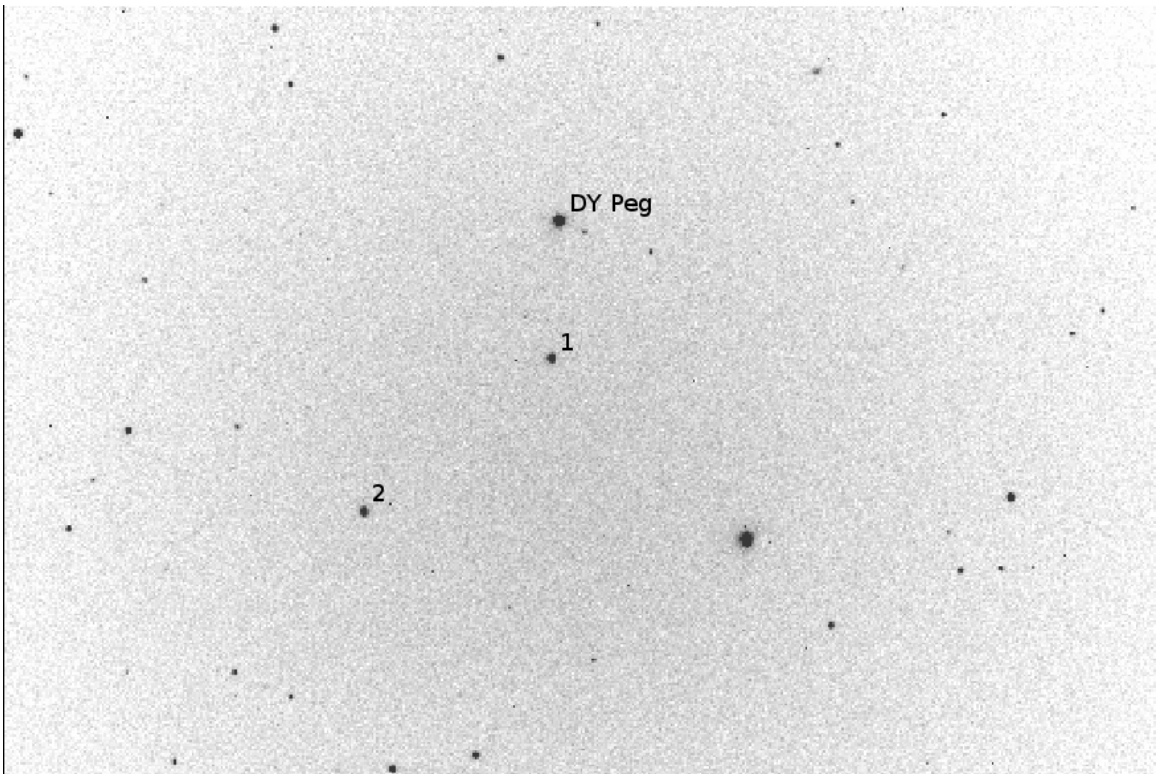

Figure 3.1: Field of DY Peg labeling both comparison stars used

the differential magnitude of a comparison star in the field of DY Pegasi ($\alpha_{2000} = 23^h08^m51\overset{s}{.}18$, $\delta_{2000} = +17°12'55\overset{''}{.}975$). The standard deviation is calculated from 3000 frames, taken during October 2007 on the WMO .31m telescope with an SBIG ST-10 XME CCD at the Cassegrain focus. The *ccdclip* algorithm did produce the smallest errors, but by such a small amount that, statistically speaking, all three values are identical. Figure 3.1 shows the field of DY Peg and the two comparison stars used. The comparison stars, labeled 1 and 2, are GSC 01712-00542 and GSC 01712-01246, respectively. The standard deviations calculated are for comparison star 1.

In trying to analyze the results of the first test, we concluded that any difference in the reduction methods may have been inadvertently washed out by using differential magnitudes. Any noise left behind from the reduction algorithm would

Table 3.2. Standard Deviation of Raw Magnitudes for Star 1 in the Field of DY Peg Using Different Reduction Algorithms (WMO .31m)

Reduction Algorithm		σ
minmax	···	0.009451
avsiglcip	···	0.009486
ccdclip	···	0.009484

have been present across the entire frame. This excess noise would have been added into the ensemble average and then canceled out during the differential magnitude determination when the ensemble average was subtracted from each individual star (Joner 2008).

In order to detect any remaining noise due to choice of reduction algorithm, we moved from differential magnitudes to raw magnitudes for our second test. This meant that we needed to calculate the extinction coefficients for each side of the pier, apply the coefficient to the raw instrumental magnitudes, and examine the frames for remaining noise. The results of this test, Table 3.2, show that the standard deviations are very similar to those found when using differential magnitudes. The *minmax* algorithm actually has the smallest σ; however, the differences between the three standard deviations are so small that they are essentially equal. Again, this seems to contradict the statements made in the IRAF help files.

After seeing the results of the second test, it was decided to try the same approach on frames from the DDT. The idea was that since the observing conditions at WMO produced frames with very little noise to begin with (an average of only 167.38 counts per pixel over the 3000 frames before processing) it would be difficult to see any major differences between the reduction algorithms. The DDT experiences a significantly greater amount of light pollution (an average of 3172.68 counts per

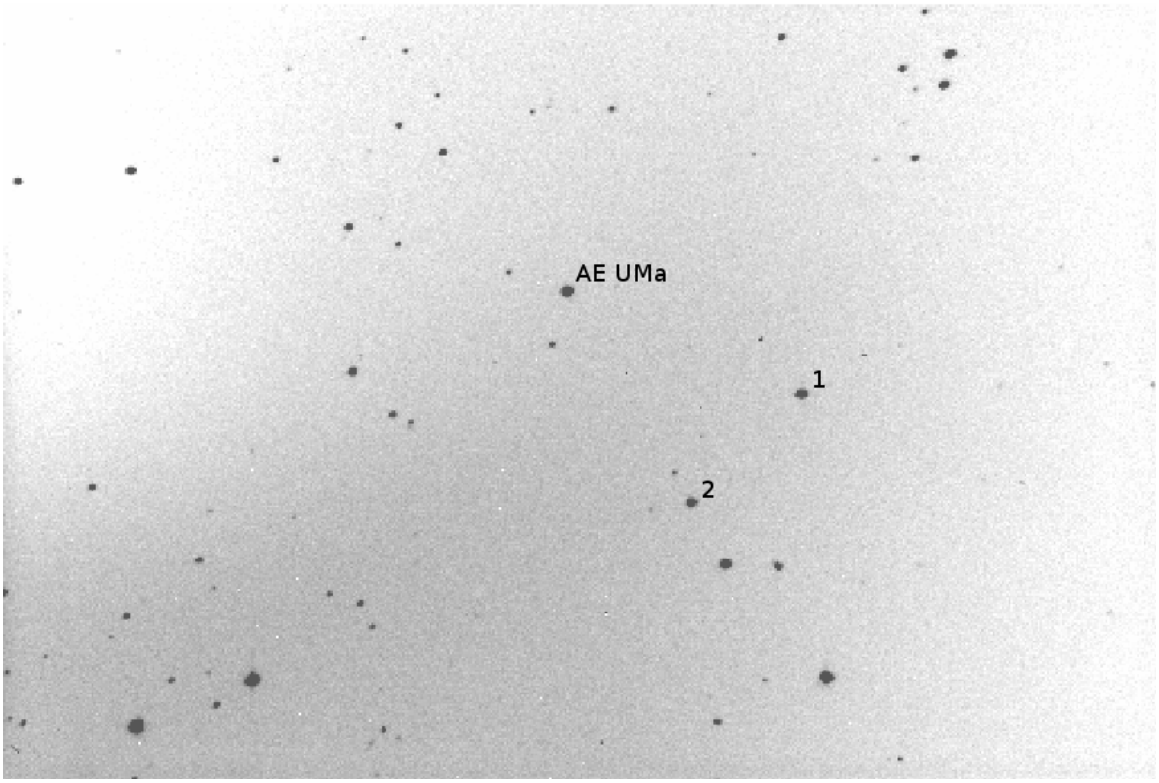

Figure 3.2: Field of AE Ursa Majoris with comparison stars labeled

pixel before processing), so any leftover noise may be more obvious on frames that are much noisier to begin with.

The same raw magnitude technique was applied to 96 frames, taken during May 2007, for a comparison star in the field of AE Ursa Majoris ($\alpha_{2000} = 09^{h}36^{m}53^{s}.1557$, $\delta_{2000} = +44°04'00''.404$). Frames were taken using the DDT .4m telescope with an SBIG ST-10 XME CCD camera at the Newtonian focus. Figure 3.2 shows the field of AE UMa with the comparison stars labeled. Star 1 is TYC 2998-1249-1 and star 2 is unidentified. Standard deviations are for star 1. Extinction curves were plotted, and the extinction coefficients were determined and applied. Unfortunately, the results disappointed once again. The results from this third attempt are shown in Table 3.3. The differences in σ are, again, statistically indistinguishable.

Table 3.3. Standard Deviation of Raw Magnitudes for Star 1 in the Field of AE UMa Different Reduction Algorithms (DDT)

Reduction Algorithm		σ
minmax	$\cdots$	0.007735
avsiglcip	$\cdots$	0.007634
ccdclip	$\cdots$	0.007492

The fact that the errors from the DDT were so small was disturbing. Because there was so much noise in the data, we were confident that differences would arise between the reduction algorithms. At this point it was finally realized that the differences were actually present, but we had been searching for them in the wrong way.

By comparing the magnitude standard deviation, whether based on differential magnitudes or raw magnitudes, the robustness of the reduction algorithm was not being tested. In the attempt to hold all other variables constant, a major component of the IRAF magnitude determination had been overlooked. We had not taken into consideration the sky fitting algorithm that calculates the sky value during the *phot* task.

IRAF determines the raw instrumental magnitude based on Eq. 3, with the *flux* being defined by Eq. 4.

$$mag = zmag - 2.5 \times log_{10}(flux) + 2.5 \times log_{10}(itime) \tag{3}$$

$$flux = sum - (area \times msky) \tag{4}$$

The value of *msky* is the average sky value surrounding the star and must be subtracted from the total number of counts in order to determine the true flux of

Table 3.4. Comparison of Sky Values and Standard Deviations (DDT)

Reduction Algorithm		Average Sky Value	Average σ	σ_s
minmax	$\cdots$	3084.65	44.35	8.855
avsiglcip	$\cdots$	3051.72	43.93	7.985
ccdclip	$\cdots$	3092.30	44.09	5.460

only the star. Regardless of which reduction algorithm was used, any remaining noise would be removed and the same value for *flux* would be returned, if the sky fitting algorithm performs well. In order to see how well the reduction algorithm performs, without being influenced by the strength of the sky fitting algorithm, we had to look at something unaffected by the sky fitting algorithm, namely, the value of the sky itself.

Table 3.4 shows, for the 96 frames taken on the DDT, the average sky value (as determined by *ap_skynoise_obhead*), the average standard deviation of the frame to frame sky value, and the standard deviation of the frame to frame sky value standard deviation (hereafter called σ_s) for the three reduction methods. As can be seen, the average sky value is very consistent, differing by less than 1.4% between the maximum and minimum values. Even the average standard deviation of the sky values is very similar for each method. However, examination of the value of σ_s shows noticeable difference between the three algorithms. The value of σ_s is a measurement of how consistent the given algorithm is at producing frames of equal "flatness" (i.e., pixels do not vary greatly across the CCD). A lower value means that it becomes more difficult to differentiate one frame from another, because they have all been reduced to a similar level of noise. Higher values mean that the differences from frame to frame are more noticeable, because each frame has not been reduced to the same noise levels.

Table 3.5. Comparison of Sky Values and Standard Deviations (WMO .31m)

Reduction Algorithm		Average Sky Value	Average σ	σ_s
minmax	···	29.24	7.497	0.4185
avsiglcip	···	28.08	7.504	0.4188
ccdclip	···	24.55	7.408	0.4383

Examining these same values for the WMO .31m gives slightly different results, but still seems to show that *ccdclip* may be the better choice. Table 3.5 shows that the average sky value for the 3,000 frames is different for the three different reduction algorithms, with *ccdclip* being the best choice. The difference of 4 - 5 counts per pixel may not seem like much, but for a star with 10,000 total counts on a poor night of seeing (4 pixel FWHM), those 4 - 5 counts can total up to over 200 counts. That is over a 2% difference in the total flux of the star, simply by using *ccdclip* as opposed to *minmax*. The strength of the sky fitting algorithm should be able to compensate for the difference, but the ultimate goal is finding the **best** reduction algorithm. The average standard deviation is essentially equal for each algorithm, as well as the values for σ_s.

These results, for both the DDT and WMO .31m, are far from conclusive. When taken in context with the statement from the *combine* help file, however, it does seems that the *ccdclip* algorithm does offer advantages for CCDs with established values for gain and readnoise. To solidify this statement, further tests are suggested on the DDT, as well as the WMO .4m and .51m telescopes.

3.2 Comparison of SIDAP and Traditional Method on DY Pegasi

As the previous section showed, SIDAP was designed to make small improvements, wherever possible, in the overall reduction process. While the gains of choosing

Table 3.6. A Comparison of Magnitudes Produced by SIDAP and Traditional Method

Method		σ
Traditional	···	0.007551
SIDAP	···	0.004398

one reduction method over another may be minimal with the CCDs currently operated at BYU, the final results of using SIDAP over the traditional reduction and photometric methods are significant.

The two stars labeled in Figure 3.1 are both used as comparison stars during differential analysis of DY Peg. A total of 1000 frames of the DY Peg field, from 9 October 2007, were reduced using the SIDAP methods, with the *ccdclip* reduction algorithm and the 95% integrated flux method in *ap_apphot*, to achieve magnitudes for both stars. The same 1000 frames were also reduced using the *minmax* reduction algorithm, and magnitudes were calculated using a 13-pixel aperture and *annulus* and a 4-pixel *dannulus*. Dr. Eric Hintz's *varstar5* program was used to produce differential magnitudes for the two comparison star ensemble.

Figure 3.3 shows a graph of the differential magnitude produced using the traditional reduction method plotted against the Heliocentric Julian Date (HJD) for GSC 01712-00542. Figure 3.4 shows the differential magnitude versus HJD as produced by SIDAP, with observations colored by pier side, plotted on the same scale as Figure 3.3. The difference in scatter between the two methods is immediately obvious. Table 3.6 shows the standard deviations of the magnitudes for both methods.

Another difference that stands out immediately is the discontinuity at the pier break. There is approximately a .005 average magnitude difference between observations on the east and west side of the pier. Observations of M67 that were reduced using SIDAP and analyzed by Dr. Benjamin Taylor reveal a significant

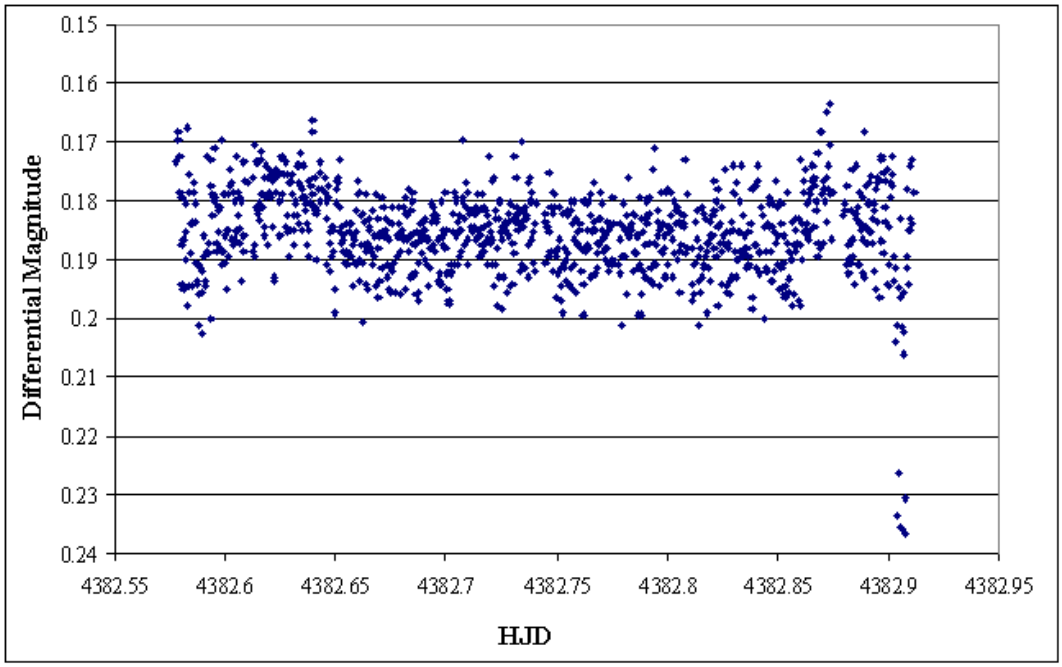

Figure 3.3: GSC 01712-00542 differential magnitude plot using traditional method.

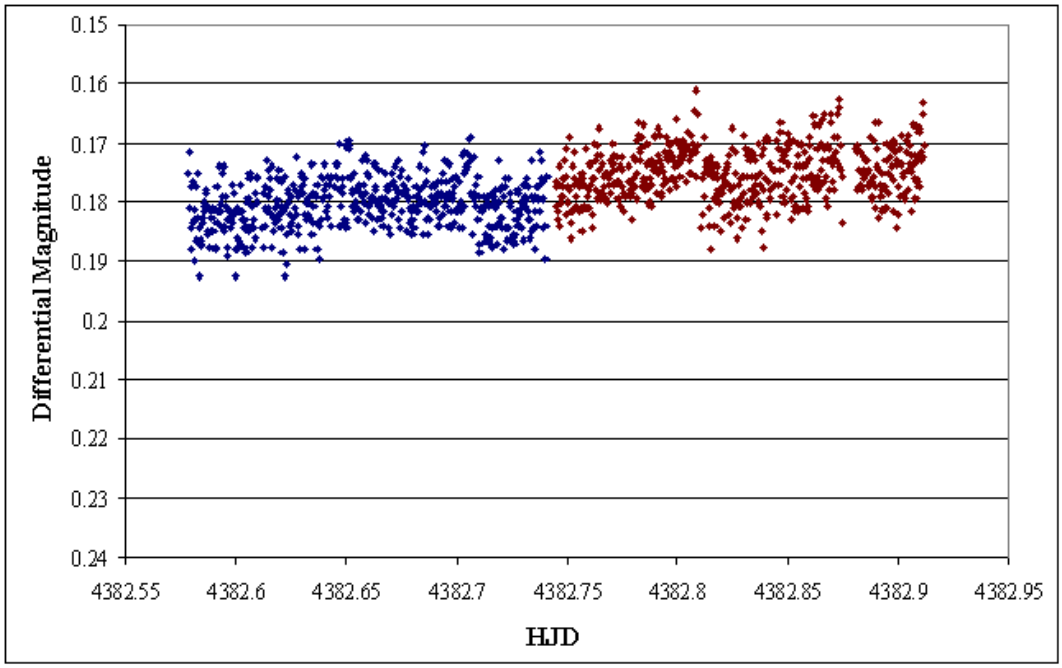

Figure 3.4: GSC 01712-00542 differential magnitude plot using SIDAP.

gradient in both the x and y dimensions across the CCD. The first results seem to indicate a .005 magnitude per 1000 pixels gradient in the x dimension and a .018 magnitude gradient per 1000 pixels in the y dimension (Joner 2008, Taylor 2008). Until a precise value for the gradient is established, it will be not be possible to perform frequency analysis on multi-periodic stars such as DY Pegasi. The presence of the gradient introduces an artificial, low-amplitude period which makes extracting the real frequencies impossible.

Despite not being able to perform any frequency analysis, times of maximum light can still be determined. Table 3.7 presents 17 times of new maximum light for DY Peg. We compared these times of maximum light to the two possible ephemerides presented by Hintz et. al (2004). Hintz et. al concluded that DY Peg was best modeled by either the second order polynomial shown in Eq. 5., or a triple linear fit given by Eq. 6, Eq. 7, and Eq. 8.

$$\mathrm{HJD}_{max} = 2429193.4464 + 0.072926383E - 2.187 \times 10^{-13}E^2 \tag{5}$$

$$\mathrm{HJD}_{max} = 2429193.4471 + 0.072926357E \tag{6}$$

$$\mathrm{HJD}_{max} = 2438276.8623 + 0.072926298E \tag{7}$$

$$\mathrm{HJD}_{max} = 2450052.1985 + 0.072926197E \tag{8}$$

The newest 17 times of maximum light were added to the existing Observed minus Calculated (O - C) plots for both the quadratic and triple linear fits. The quadratic *0* - *C*, Figure 3.5, shows that the new observed maxima (shown in orange) consistently precede the calculated times. Figure 3.6 shows the O - C for the triple linear fit. As with the quadratic fit, the observed maxima (shown in orange) consistently precede the calculated maxima. The triple linear fit is a better fit to the

Table 3.7. New Times of Maximum Light for DY Peg

HJD 2450000.0+	Quadratic Cycle Number	Triple Linear Cycle Number	HJD 2450000.0+	Quadratic Cycle Number	Triple Linear Cycle Number
4374.60662	345296	59271	4382.62855	345406	59381
4374.67935	345297	59272	4382.70135	345407	59382
4374.75240	345298	59273	4382.77377	345408	59383
4374.82555	345299	59274	4382.84714	345409	59384
4374.89830	345300	59275	4383.64927	345420	59395
4376.64848	345324	59299	4383.72222	345421	59396
4376.72096	345325	59300	4383.79467	345422	59397
4376.79356	345326	59301	4383.86817	345423	59398
4376.86757	345327	59302	...	...	...

new data points, but we cannot say whether it is correct. It is possible that Hintz et. al presented a period that was too short, but it is equally likely that DY Peg has undergone another period break.

Once the gradient on the WMO .31m is better established, we will be able to reduce archival data and reexamine DY Peg with frequency analysis to determine whether another period break has occurred. Until that time, we can only say that the triple linear fit presented by Hintz et. al is the best fit.

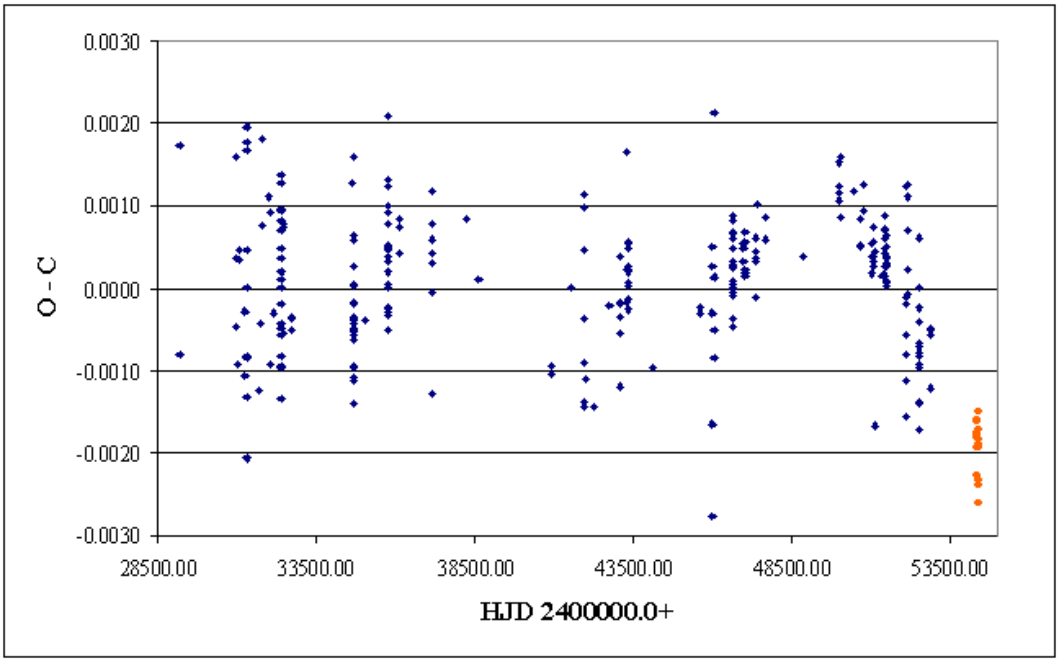

Figure 3.5: *O* - *C* diagram for quadratic fit as given by Eq. 5.

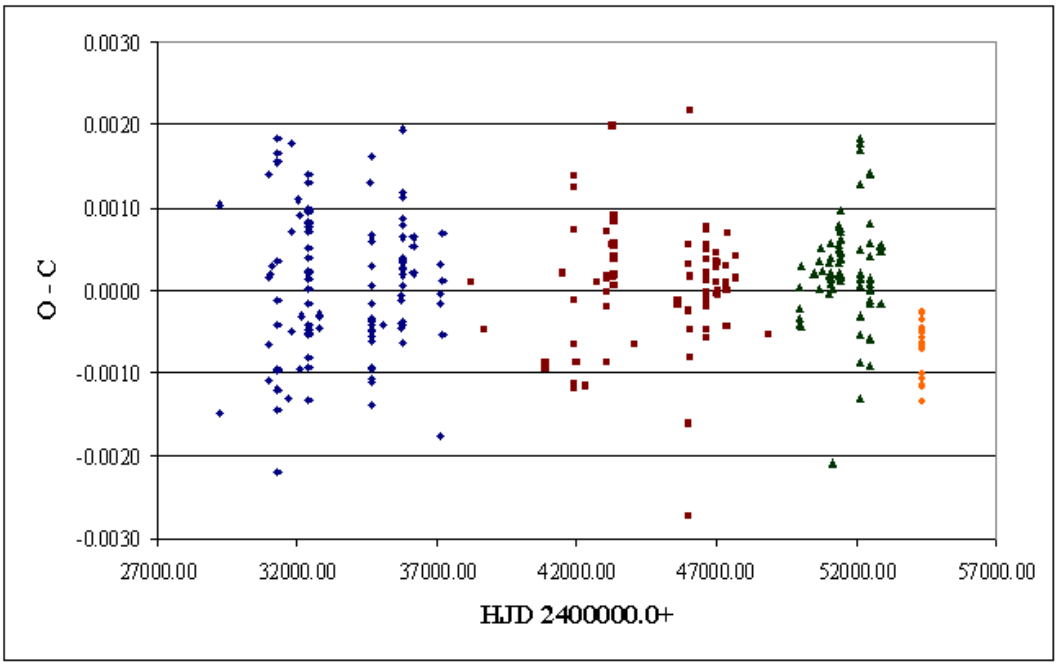

Figure 3.6: *O* - *C* diagram for triple linear fit as given by Eq. 6, Eq. 7, and Eq. 8.

Chapter 4

Conclusions and Suggestions

Creating SIDAP has been a long and arduous process, the results of which are just now being fully understood. The speed at which data can be reduced because of automation allows for the exploration of previously unused and unknown IRAF tasks, such as *psfmeasure*. The exploration of these new tasks has led to significant improvements in the aperture photometry techniques being used at BYU.

The development of the *ap_apphot* script may be the most important advancement. The idea of using differential aperture sizes across a single frame, and throughout the observation run, is something that has already yielded huge benefits in the precision with which stellar magnitudes are calculated. It has also led to the discovery of previously undiscovered gradients on the WMO .31m and .4m telescopes. Without the increase in precision, these gradients may have continued to go unnoticed.

The techniques used as part of SIDAP are still in a fledgling state and will continue to undergo major changes and improvements. Part of the planned improvements include rewriting all of the scripts using a more uniform style of code, including better comments. We also wish to write them in such a way as to render them "readable" to other astronomy students with no programming experience. We will also be removing unnecessary parts of scripts, such as the integrated FWHM option from the *ap_apphot* script. It was programmed before we discovered the integrated flux method and has since been used solely for comparison purposes. By removing these unnecessary or unused parts of the scripts, we will make them run more efficiently and remove any possible confusion that may arise.

As we have distributed the scripts to other individuals, several small problems have arisen that we have currently resolved simply through brute force. Most of these problems have been contingencies that we had simply not anticipated. As we rewrite

the scripts, we will search for more permanent solutions in hopes of making the scripts "smart enough" to overcome the majority of situations that may arise.

Future plans also include modifying the current IRAF *psfmeasure* task to allow for errors without breaking out of the script. If *psfmeasure* is unable to calculate a point-spread-function for the object in question, it currently breaks out of the script and crashes. We hope to change this and instead have it simply notify the user of the problem, mark the star for inspection, and continue running. The *psfmeasure* task also has the ability of marking stars as being saturated. We would like to take advantage of this ability, but are currently unable to do so because of the way that *psfmeasure* deals with the saturated stars. We hope to also rewrite *psfmeasure* to simply notify the user of a saturated star so that the star won't be used as a comparison star during analysis.

Though there will be many significant changes to SIDAP in the near future, the basic principles explained here and in Iverson (2008) will remain. We hope that by making these scripts available to others we can improve the overall quality of research performed at BYU, and also help introduce new students to the ideas of IRAF automation in hopes that they will continue what we have begun.

References

Da Costa, G. S. 1992, ASP Conference Series, 23, 90

Hintz, E. G., Joner, M. D., Ivanushkina, M., & Pilachowski, C. A. 2004, PASP, 116, 543

Iverson, P. 2008, Senior Thesis

Joner, M. 2008, private communications

Ofek, Eran 2008, (http://wise-obs.tau.ac.il/∼eran/iraf/index.html)

Taylor, B. 2008, private communications

Valdez, F. 2007, private communications

Appendix A

Automated IRAF Scripts

A.1 SIDAP Scripts

A.1.1 *ap_rfits*

```
#DOCUMENTATION: ap_rfits_v_2.cl
#           ap_rfits_v_2.cl by Paul Iverson
#
#           ap_rfits_v_2.cl is a script designed to utilized and facilitate IRAF's RFITS command.
#
#           This script performs essentially the same operation as the IRAF command with some added
#           functionality. Specifically, the ap_rfits script locates and sorts raw object, flat, bias, and dark
#           frames based on a set of parameters. Flat frames are further sorted by filter. The ap_rfits script
#           then performs the rfits command using a User-defined name for object files.
#           Flat, bias, and dark frames are named flat{filter}, zero, and dark respectively.
#
#           IRAF packages required
#                  system
#                  dataio
#                  imutil
#                  language
#
#           Linux commands required
#                  mkdir
#                  date
#
#           Possible sources of error:
#                  Inaccurate entries in filename, imagetype, or filter entries in header -> no data in
#                  datalists

procedure ap_rfits_v_2 (object_name)

string object_name {prompt = "Enter desired name for object files"}

begin
    string objectname
    objectname = object_name
end

#variable declarations
int         n=1
string      s1=""
string      s2=""
string      s3=""
string      files=""
string      filter=""
string      checkfilter=""
string      datestring=""
string      timestring=""
struct      *list1

#deletes possibly incompatible datalists
del data*

#cleans tmp folder
del /tmp/iraf*.fits

#extracts filename, imagetyp, and filter from raw frames
hselect ("*.{fit}","$I,IMAGETYP,FILTER",yes,>"datalist1")

sections @datalist1
if (sections.nimages>0){
     #filters raw object frames from flat, bias, and dark raw frames
     match ("{light}","datalist1",>>"datalistlightA")
```

```
     sections @datalistlightA
     if (sections.nimages==0){
           match ("{object}","datalist1",>>"datalistlightA")
     }
     ;

     match ("{flat}","datalistlightA",stop+,>>"datalistlightB")
     match ("{bias}","datalistlightB",stop+,>>"datalistlightC")
     match ("{zero}","datalistlightC",stop+,>>"datalistlightD")
     match ("{dark}","datalistlightD",stop+,>>"datalistlightfiles")

     #rfits raw object frames using User-defined name
     sections @datalistlightfiles
     if (sections.nimages>0){
           for (n=1;n<=sections.nimages;n+=1){
                 tabpar ("datalistlightfiles",1,n)
                 files=tabpar.value
                 rfits (files,"","obj-"//(objectname)//"-"//n)
           }
     }
     ;
}
;
del data*

#extracts filename, imagetyp, and filter from raw frames
hselect ("*.{fit}","$I,FILTER",yes,>"datalist1")

sections @datalist1
if (sections.nimages>0){
     #filters raw flat frames from object, bias, and dark raw frames
     match ("{flat}","datalist1",>>"datalistflatfiles")

     #rfits raw flat frames by filter
     sections @datalistflatfiles
     if (sections.nimages>0){
           tsort ("datalistflatfiles",2)
           list1="datalistflatfiles"
           while (fscan(list1,s1,s2)!=EOF){
                 files=(s1)
                 filter=(s2)

                 if (filter!=checkfilter){
                       n=1
                 }
                 ;

                 rfits (files,"","flat"//filter//"-"//n)
                 n=n+1

                 checkfilter=filter
           }
           ;
     }
     ;
}
;
del data*

#extracts filename and imagetyp from raw frames
hselect ("*.{fit}","$I,IMAGETYP",yes,>"datalist1")

sections @datalist1
if (sections.nimages>0){
     #filters raw bias frames from object, flat, and dark raw frames
     match ("{bias}","datalist1",>>"datalistzerofiles")
     sections @datalistzerofiles
     if (sections.nimages==0){
           match ("{zero}","datalist1",>>"datalistzerofiles")
     }
     ;

     #rfits raw bias frames
     sections @datalistzerofiles
     if (sections.nimages>0){
           for (n=1;n<=sections.nimages;n+=1){
                 tabpar ("datalistzerofiles",1,n)
```

```
                files=tabpar.value
                imgets (files,param="EXPTIME")
                if (real(imgets.value)==0){
                      rfits (files,"","zero-"//n)
                }
                ;
          }
     }
     ;

     #filters raw dark frames from object, flat, and bias raw frames
     match ("{dark}","datalist1",>> "datalistdarkfiles")

     #rfits raw dark frames
     sections @datalistdarkfiles
     if (sections.nimages>0){
          for (n=1;n<=sections.nimages;n+=1){
                tabpar ("datalistdarkfiles",1,n)
                files=tabpar.value
                imgets (files,param="EXPTIME")
                if (real(imgets.value)>0){
                      rfits (files,"","dark-"//n)
                }
                ;
          }
     }
     ;
}
;

#copies all .fit files to filesrawfit folder
if (access("filesrawfit")){
     cd filesrawfit
     del *
     cd ..
     mv *.fit filesrawfit
     mv *.FIT filesrawfit
}
;

if (access("filesrawfit")!=yes){
     mkdir filesrawfit
     mv *.fit filesrawfit
     mv *.FIT filesrawfit
}
;

#deletes possibly incompatible datalists
del data*

#adds date and time of when script was applied to frames to headers
date '+DATE:%m/%d/%y%nTIME:%H:%M:%S' > "datalist"
tabpar ("datalist",1,1)
datestring=tabpar.value
tabpar ("datalist",1,2)
timestring=tabpar.value

hedit ("*.fits","AP_RFITS",datestring//" "//timestring,add+,ver-)

#deletes possibly incompatible datalists
del data*

beep
```

A.1.2 *ap_calhead*

```
#DOCUMENTATION: ap_calhead_v_2.cl
#           ap_calhead_v_2.cl by Paul Iverson
#
#           ap_calhead_v_2.cl is a script designed to modify the headers of calibration frames
#           rfits'ed with the   ap_rfits_v_2.cl script.
#
#           This script packages the initial header changes performed previously either by hand or
#           using cmds files.
#           Specifically, it modifies SUBSET, IMAGETYP, and OBJECT.
#
#           IRAF packages required
#                 imutil
#
#           Linux commands required
#                 date
#
#           Possible sources of error:
#                 Inaccurate filenames or filter entries in header -> no data in datalists

#variable declarations
string      s1=""
string      s2=""
string      files=""
string      filter=""
string      datestring=""
string      timestring=""
struct      *list1

#deletes possibly incompatible datalists
del data*

#extracts filename and filter from rfits'ed flat frames
hselect ("*flat*","$I,FILTER",yes,>"datalist1")
list1="datalist1"
while (fscan(list1,s1,s2)!=EOF){
     files=(s1)
     filter=(s2)

     #edits header field "SUBSET" for all flat frames
     hedit (files,"SUBSET",filter,add+,ver-)

     #edits header field "IMAGETYP" for all flat frames
     hedit (files,"IMAGETYP","flat",add+,ver-)

     #edits header field "OBJECT" for all flat frames
     hedit (files,"OBJECT","Flat "//filter,add+,ver-)
}

#edits header field "IMAGETYPE" for all bias and dark frames
hedit ("*zero*.fits","IMAGETYP","zero",add+,ver-)
hedit ("*dark*.fits","IMAGETYP","dark",add+,ver-)

#edits header field "OBJECT" for all bias and dark frames
hedit ("*zero*.fits","OBJECT","Zero",add+,ver-)
hedit ("*dark*.fits","OBJECT","Dark",add+,ver-)

#deletes possibly incompatible datalists
del data*

#adds date and time of when script was applied to frames to headers
date '+DATE:%m/%d/%y%nTIME:%H:%M:%S' > "datalist"
tabpar ("datalist",1,1)
datestring=tabpar.value
tabpar ("datalist",1,2)
timestring=tabpar.value

hedit ("flat*.fits","AP_CALHEAD",datestring//" "//timestring,add+,ver-)
hedit ("zero*.fits","AP_CALHEAD",datestring//" "//timestring,add+,ver-)
hedit ("dark*.fits","AP_CALHEAD",datestring//" "//timestring,add+,ver-)

#deletes possibly incompatible datalists
del data*

beep
```

A.1.3 *(∗)_obhead*

```
#DOCUMENTATION: wmo_12_obhead_v_2.cl
#    wmo_12_obhead_v_2.cl by Paul Iverson
#
#           wmo_12_obhead_v_2.cl is a script designed to modify the headers of object
#           frames taken on WMO's 12 in. and rfits'ed with the ap_rfits_v_2.cl script.
#
#           This script packages the initial header changes performed previously either by
#           hand or using cmds files. Specifically, it modifies TELESCOP, SUBSET,
#           IMAGETYP, OBSERVER, OBSERVAT, RA, DEC, UT, AIRMASS, and HJD.
#
#           IRAF packages required
#                  astutil
#                  imutil
#
#           Linux commands required
#                  sed
#                  echo
#                  date
#
#           Possible sources of error:
#                  Inaccurate filenames or filter entries in header files -> no data in
#                  datalists

procedure wmo_12_obhead_v_2 (telescope,object_RA,object_DEC,hour_correct,hour_change)

string telescope {prompt="Enter the diameter of the telescope",enum="12|16|20|36"}
string object_RA {prompt="Enter the RA of your object (i.e. 12 34 54.5)"}
string object_DEC {prompt="Enter the Dec of your object (i.e. +12 34 54.5)"}
string hour_correct {prompt="Enter the time reference of the image",enum="Local|UT"}
string hour_change {prompt="Enter the time difference from Greenwich UT on the date of image"}

begin
     string tele
     string RAvalue
     string DECvalue
     string hr_corr
     string hr_change
end

#variable declarations
int  n=1
int  l=1
real time
real checktime
string      RAcheck=""
string      DECcheck=""
string      s1=""
string      s2=""
string      s3=""
string      s4=""
string      s5=""
string      s6=""
string      files=""
string      timedate=""
string      timeans=""
string      UTnow=""
string      datestring=""
string      timestring=""
struct      *list1
struct      *list2

#deletes possibly incompatible datalists
del data*

#adds date and time of when script was applied to frames to headers
date '+DATE:%m/%d/%y%nTIME:%H:%M:%S' > "datalist"
tabpar ("datalist",1,1)
datestring=tabpar.value
tabpar ("datalist",1,2)
timestring=tabpar.value

tele=telescope

hedit ("obj*.fits","WMO_"//tele//"_OBHEAD",datestring//" "//timestring,add+,ver-)

#deletes possibly incompatible datalists
del data*
```

```
#edits header field "TELESCOP"
hedit ("*obj*.fits","TELESCOP",tele//"-inch",add+,ver-)

#edits header field "SUBSET"
hedit ("*obj*.fits","SUBSET","(FILTER)",add+,ver-)

#edits header field "IMAGETYP"
hedit ("*obj*.fits","IMAGETYP","object",add+,ver-)

#edits header field "OBSERVER"
hedit ("*obj*.fits","OBSERVER","BYU-WMO",add+,ver-)

#edits header field "OBSERVAT"
hedit ("*obj*.fits","OBSERVAT","wmo",add+,ver-)

#edits header fields "RA" and "DEC" and modifies for use
sections ("*obj*.fits",opt="fullname",>"datalist1")
list1="datalist1"
while (fscan(list1,s1)!=EOF){
     files=(s1)

     imgets (files,param="RA")
     RAcheck=(imgets.value)
     if (RAcheck=="0"){
            imgets (files,param="OBJCTRA")
            RAcheck=(imgets.value)
            if (RAcheck!="0"){
                  hedit (files,"RA","(OBJCTRA)",add+,ver-)
                  hedit (files,"RA","(OBJCTRA)",add+,ver-)
            }
            ;
            if (RAcheck=="0"){
                  clear
                  RAvalue=object_RA

                  hedit ("*obj*.fits","RA",(RAvalue),add+,ver-)
                  hedit ("*obj*.fits","RA",(RAvalue),add+,ver-)
            }
            ;
     }
     ;
     imgets (files,param="DEC")
     DECcheck=(imgets.value)
     if (DECcheck=="0"){
            imgets (files,param="OBJCTDEC")
            DECcheck=(imgets.value)
            if (DECcheck!="0"){
                  hedit (files,"DEC","(OBJCTDEC)",add+,ver-)
                  hedit (files,"DEC","(OBJCTDEC)",add+,ver-)
            }
            ;
            if (DECcheck=="0"){
                  clear
                  DECvalue=object_DEC

                  hedit ("*obj*.fits","DEC",(DECvalue),add+,ver-)
                  hedit ("*obj*.fits","DEC",(DECvalue),add+,ver-)
            }
            ;
     }
     ;
}
del data*

hselect ("*obj*.fits","$I,RA,DEC",yes,>"datalist1")
sed "s/\ /:/g" "datalist1" > "datalist2"
list1="datalist2"
while (fscan(list1,s1,s2,s3)!=EOF){
     hedit (s1,"RA",s2,add+,ver-)
     hedit (s1,"DEC",s3,add+,ver-)
}
del data*

#edits header field "EPOCH"
hedit ("*obj*.fits","EPOCH","2000.0",add+,ver-)
```

```
#edits header field "UT"
hselect ("*obj*.fits","$I,DATE-OBS",yes,>"datalist1")
sections @datalist1
if (sections.nimages>0){
     match ("T","datalist1",stop-,>"datalistdateobs")
     match ("T","datalist1",stop+,>"datalisttimeobs")
     sections @datalistdateobs
     if (sections.nimages>0){
            sed "s/[0-9-]*[T]//g" "datalistdateobs" > "datalist2"
            list1="datalist2"
            while (fscan(list1,s1,s2)!=EOF) {
                  hedit (s1,"UT",s2,add+,ver-)
                  hedit (s1,"TIME-DAT","DATE-OBS",add+,ver-)
            }
     }
     ;
     sections @datalisttimeobs
     if (sections.nimages>0){
            hselect ("*obj*.fits","$I,TIME-OBS",yes,>"datalist3")
            sections @datalist3
            if (sections.nimages>0){
                  list1="datalist3"
                  while (fscan(list1,s1,s2)!=EOF) {
                        hedit (s1,"UT",s2,add+,ver-)
                        hedit (s1,"TIME-DAT","TIME-OBS",add+,ver-)
                  }
            }
            ;
     }
     ;
}
;
del data*

#determines sidereal time and edits header field "ST"
!echo "st = mst(@'DATE-OBS', UT, obsdb (observat, \"longitude\"))" > st.cmds
asthedit ("*obj*.fits","st.cmds",table="",verbose+)
del st.cmds

#determines airmass and edits header field "AIRMASS"
setairmass ("*obj*.fits")

#determines julian date, helian julian date, etc. and edits header fields "JD", etc.
setjd ("*obj*.fits")

#checks for incorrect start times (i.e. UT set as Local or set as actual UT)
hselect ("*obj*.fits","$I,TIME-DAT",yes,>"datalist1")
list1="datalist1"
while (fscan(list1,s1,s2)!=EOF){
     files=(s1)
     timedate=(s2)

     if (timedate=="DATE-OBS"&&n==1){
            hselect (files,"DATE-OBS,UT,ST,RA",yes,>"datalist2")
            list2="datalist2"
            while (fscan(list2,s3,s4,s5,s6)!=EOF){
                  UTnow=(s4)
                  clear
                  print ("DATE-OBS: ",s3)
                  print ("")
                  print ("Given UT: ",s4)
                  print ("Calculated ST: ",s5)
                  print ("Object RA: ",s6)
                  print ("")
                  hr_corr=hour_correct
                  print ("")
            }

            if (hr_corr=="Local"){
                  hr_change=hour_change
                  time=real(UTnow)
                  time=time+real(hr_change)
                  if (time>24){
                        time=time-24
                  }
                  ;
                  hedit (files,"UT",time,add+,ver-)
```

```
                hedit (files,"TIME-DAT","UT Corrected",add+,ver-)

                #determines sidereal time and edits header field "ST"
                !echo "st = mst(@'DATE-OBS', UT, obsdb (observat, \"longitude\"))" > st.cmds
                asthedit (files,"st.cmds",table="",verbose+)
                del st.cmds

                #determines airmass and edits header field "AIRMASS"
                setairmass (files)

                #determines julian date, helian julian date, etc. and edits header fields "JD", etc.
                setjd (files)
        }
        ;
        del datalist2
}
;
if (timedate=="DATE-OBS"&&n!=1&&hr_corr=="Local"){
        hselect (files,"UT",yes,>"datalist2")
        list2="datalist2"
        while (fscan(list2,s3)!=EOF){
                time=real(s3)
                time=time+real(hr_change)
                if (time>24){
                        time=time-24
                }
                ;
                hedit (files,"UT",time,add+,ver-)

                #determines sidereal time and edits header field "ST"
                !echo "st = mst(@'DATE-OBS', UT, obsdb (observat, \"longitude\"))" > st.cmds
                asthedit (files,"st.cmds",table="",verbose+)
                del st.cmds

                #determines airmass and edits header field "AIRMASS"
                setairmass (files)

                #determines julian date, helian julian date, etc. and edits header fields "JD", etc.
                setjd (files)

                hedit (files,"TIME-DAT","UT Corrected",add+,ver-)
        }
        del datalist2
}
;
n=n+1

if (timedate=="TIME-OBS"&&l==1){
        hselect (files,"DATE-OBS,TIME-OBS,UT,ST",yes,>"datalist2")
        list2="datalist2"
        while (fscan(list2,s3,s4,s5,s6)!=EOF){
                UTnow=(s5)
                clear
                print ("DATE-OBS: ",s3)
                print ("TIME-OBS: ",s4)
                print ("")
                print ("Given UT: ",s5)
                print ("Calculated ST: ",s6)
                print ("")
                hr_corr=hour_correct
                print ("")
        }

        if (hr_corr=="Local"){
                hr_change=hour_change
                time=real(UTnow)
                time=time+real(hr_change)
                if (time>24){
                        time=time-24
                }
                ;
                hedit (files,"UT",time,add+,ver-)

                #determines sidereal time and edits header field "ST"
                !echo "st = mst(@'DATE-OBS', UT, obsdb (observat, \"longitude\"))" > st.cmds
                asthedit (files,"st.cmds",table="",verbose+)
                del st.cmds
```

```
                #determines airmass and edits header field "AIRMASS"
                setairmass (files)

                #determines julian date, helian julian date, etc. and edits header fields "JD", etc.
                setjd (files)
          }
          ;
          hedit (files,"TIME-DAT","UT Corrected",add+,ver-)
          del datalist2
     }
     ;
     if (timedate=="TIME-OBS"&&l!=1&&hr_corr=="Local"){
          hselect (files,"UT",yes,>"datalist2")
          list2="datalist2"
          while (fscan(list2,s3)!=EOF){
                time=real(s3)
                time=time+real(hr_change)
                if (time>24){
                      time=time-24
                }
                ;
                hedit (files,"UT",time,add+,ver-)

                #determines sidereal time and edits header field "ST"
                !echo "st = mst(@'DATE-OBS', UT, obsdb (observat, \"longitude\"))" > st.cmds
                asthedit (files,"st.cmds",table="",verbose+)
                del st.cmds

                #determines airmass and edits header field "AIRMASS"
                setairmass (files)

                #determines julian date, helian julian date, etc. and edits header fields "JD", etc.
                setjd (files)

                hedit (files,"TIME-DAT","UT Corrected",add+,ver-)
          }
          del datalist2
     }
     ;
     l=l+1

     if (hr_corr=="UT"){
          hedit (files,"TIME-DAT","UT Given",add+,ver-)
     }
     ;
}

#deletes possibly incompatible datalists
del data*

beep
```

A.1.4 *ap_gain_rdnoise_(*)

ap_findgain

```
# ap_findgain.cl by Paul Iverson modified from findgain.cl script from IRAF package
#
#    - calculates the gain and readnoise given two flats and two
#    bias frames.  Algorithm (method of Janesick) courtesy Phil Massey.
#
#    flatdif = flat1 - flat2
#    biasdif = bias1 - bias2
#
#    e_per_adu = ((mean(flat1)+mean(flat2)) - (mean(bias1)+mean(bias2))) /
#                ((rms(flatdif))**2 - (rms(biasdif))**2)
#
#    readnoise = e_per_adu * rms(biasdif) / sqrt(2)
#

procedure findgain (flat1, flat2, zero1, zero2)

string      flat1                       {prompt="First flat frame"}
string      flat2                       {prompt="Second flat frame"}
string      zero1                       {prompt="First zero frame"}
string      zero2                       {prompt="Second zero frame"}

string      section = ""                {prompt="Selected image section"}
string      center = "mean"             {prompt="Central statistical measure", enum="mean|midpt|mode"}
int         nclip = 3                         {prompt="Number of clipping iterations"}
real lsigma = 4                 {prompt="Lower clipping sigma factor"}
real usigma = 4                 {prompt="Upper clipping sigma factor"}
real binwidth = 0.1             {prompt="Bin width of histogram in sigma"}
bool verbose = no               {prompt="Verbose output?"}

begin
     string f1, f2, z1, z2
     string lf1, lf2, lz1, lz2
     string flatdif, zerodif

     #flatdif = mktemp ("tmp$iraf")
     #zerodif = mktemp ("tmp$iraf")

     f1 = flat1
     f2 = flat2
     z1 = zero1
     z2 = zero2

     lf1 = f1//section
     lf2 = f2//section
     lz1 = z1//section
     lz2 = z2//section

     imarith (lf1, "-", lf2, "flatdif")
     imarith (lz1, "-", lz2, "zerodif")
end

int         n=1
real e_per_adu, readnoise, m_f1, m_f2, m_b1, m_b2, s_fd, s_bd, junk
string      s1=""
string      s2=""
struct      *list1
struct      images

#printf ("%s,%s,%s,%s,%s,%s\n", lf1, lf2, lz1, lz2, "flatdif", "zerodif") | scan (images)

imstat
(lf1,fields=center//",stddev",lower=INDEF,upper=INDEF,nclip=nclip,lsigma=lsigma,usigma=usigma,binwidth=binwidth
,format-,>> "statsfile")
imstat
(lf2,fields=center//",stddev",lower=INDEF,upper=INDEF,nclip=nclip,lsigma=lsigma,usigma=usigma,binwidth=binwidth
,format-,>> "statsfile")
imstat
(lz1,fields=center//",stddev",lower=INDEF,upper=INDEF,nclip=nclip,lsigma=lsigma,usigma=usigma,binwidth=binwidth
,format-,>> "statsfile")
imstat
(lz2,fields=center//",stddev",lower=INDEF,upper=INDEF,nclip=nclip,lsigma=lsigma,usigma=usigma,binwidth=binwidth
,format-,>> "statsfile")
imstat
("flatdif.fits",fields=center//",stddev",lower=INDEF,upper=INDEF,nclip=nclip,lsigma=lsigma,usigma=usigma,binwid
th=binwidth,format-,>> "statsfile")
```

```
imstat
("zerodif.fits",fields=center//",stddev",lower=INDEF,upper=INDEF,nclip=nclip,lsigma=lsigma,usigma=usigma,binwid
th=binwidth,format-,>> "statsfile")

del ("flatdif*", verify-)
del ("zerodif*", verify-)

list1 = "statsfile"
while (fscan(list1,s1,s2) != EOF){
     if (n == 1){
          m_f1 = real(s1)
     }
     ;
     if (n == 2){
          m_f2 = real(s1)
     }
     ;
     if (n == 3){
          m_b1 = real(s1)
     }
     ;
     if (n == 4){
          m_b2 = real(s1)
     }
     ;
     if (n == 5){
          s_fd = real(s2)
     }
     ;
     if (n == 6){
          s_bd = real(s2)
     }
     ;
     n=n+1
}

e_per_adu = ((m_f1 + m_f2) - (m_b1 + m_b2)) / (s_fd**2 - s_bd**2)
readnoise = e_per_adu * s_bd / sqrt(2)

# round to three decimal places
e_per_adu = real (nint (e_per_adu * 1000.)) / 1000.
readnoise = real (nint (readnoise * 1000.)) / 1000.

# print results
if (verbose == yes) {
     printf ("FINDGAIN:\n")
     printf ("  center = %s, binwidth = %g\n", center, binwidth)
     printf ("  nclip = %d, lsigma = %g, usigma = %g\n",nclip, lsigma, usigma)
     printf ("\n  Flats      = %s  &  %s\n", lf1, lf2)
     printf ("  Zeros      = %s  &  %s\n", lz1, lz2)
     printf ("  Gain       = %5.2f electrons per ADU\n", e_per_adu)
     printf ("  Read noise = %5.2f electrons\n", readnoise)
}
;

if (verbose == no){
     printf ("%5.2f\t%5.2f\n", e_per_adu, readnoise)
}
;

del stats*
del /tmp/*.fits
```

ap_gain_rdnoise_collector

```
#DOCUMENTATION: ap_gain_rdnoise_collector_v_4.cl
#    ap_gain_rdnoise_collector_v_4.cl by Paul Iverson
#
#          ap_gain_rdnoise_collector_v_4.cl is a script designed to collect gain and
#          rdnoise data from raw bias and flat frames.
#
#          IRAF packages required
#                imutil
#
#          Possible sources of error:
#                Inaccurate filenames or filter entries in header -> no data in
#                      datalists

procedure ap_gain_rdnoise_collector_v_4 (frame_num)

string frame_num         {prompt="Enter the number of frames to use"}

begin
     string framenum
end

#variable declarations
int         n=1
int         t=1
int         tn
int         f1,f2,fn
int         b1,b2,bn
int         xlen,ylen
int         x1,x2,x3,x4,x5,x6
int         y1,y2,y3,y4,y5,y6
int         numtotal,numflat,numbias
int         nflat, nbias
int         nfactflat, nfactbias, remaining
string      s1=""
string      s2=""
string      s3=""
string      s4=""
string      files=""
string      filter=""
string      checkfilter=""
string      flat1,flat2
string      bias1,bias2
string      temp
struct      *list1
struct      *list2

#deletes possibly incompatible datalists
del data*

#extracts filename and filter from raw flat frames
hselect ("*{flat}*.{fit}","$I,FILTER",yes,>"datalistflat")

#creates flat table composed of filter and number; creates datalist of flat frames to be used
sections @datalistflat
numtotal=sections.nimages
if (numtotal>0){
     tsort ("datalistflat",2)
     list1="datalistflat"
     while (fscan(list1,s1,s2)!=EOF){
           filter=(s2)

           if (filter!=checkfilter&&numtotal>0){
                 print (filter,>>"datalistfiltertable")

                 clear
                 match (filter//"$","datalistflat",>"datalistflat"//filter)
                 sections ("@datalistflat"//filter)
                 numflat=sections.nimages

                 print ("")
                 print ("Total number of flats remaining: ",numtotal)
                 print ("Number of flat ",filter," frames: ",numflat)
                 print ("")
                 framenum=frame_num
                 while (int(framenum)>numflat){
                       framenum=frame_num
                 }
```

```
                print (framenum,>>"datalistfilternumtouse")

                numtotal=numtotal-numflat
            }
            ;

            checkfilter=filter
     }
}
;
joinlines ("datalistfiltertable,datalistfilternumtouse",>>"datalistfilternumtable"

list1="datalistfilternumtable"
while (fscan(list1,s1,s2)!=EOF){
     filter=(s1)
     numflat=int(s2)

     list2="datalistflat"//filter
     while (fscan(list2,s3,s4)!=EOF){
            files=(s3)

            if (n<=numflat){
                print (files,>>"datalistflatuse")
            }
            ;
            n=n+1
     }
     n=1
}

#resets counter to 1
n=1

#extracts filename and imagetyp from raw frames
hselect ("*{bias}*.{fit}","$I,IMAGETYP",yes,>"datalistbias")

#creates datalist with bias frames to be used
sections @datalistbias
if (sections.nimages==0){
     del datalistbias
     hselect ("*{zero}*.{fit}","$I,IMAGETYP",yes,>"datalistbias")
}
;
sections @datalistbias
numbias=sections.nimages
if (numbias>0){
     print ("")
     print ("Number of bias frames: ",numbias)
     print ("")
     framenum=frame_num
     while (int(framenum)>numbias){
            framenum=frame_num
     }

     list1="datalistbias"
     while (fscan(list1,s1,s2)!=EOF){
            files=(s1)

            if (n<=int(framenum)){
                print (files,>>"datalistbiasuse")
                n=n+1
            }
            ;
     }
}
;
n=1

numbias=int(framenum)

clear

#prints to screen results of compiled flat and bias tables
list1="datalistfilternumtable"
while (fscan(list1,s1,s2)!=EOF){
     print ("Using ",s2," flat ",s1," frames")
}
```

```
print ("Using ",numbias," bias frames")

sleep (2)

#recalculates number of flat and bias frames to be used
sections @datalistflatuse
numflat=sections.nimages
sections @datalistbiasuse
numbias=sections.nimages

if (numflat>0&&numbias>0) {
     nflat = numflat
     nbias = numbias

     nfactflat = (nflat * (nflat - 1)) / 2
     nfactbias = (nbias * (nbias - 1)) / 2

     remaining = nfactflat * nfactbias
}
;

#determines gain and rdnoise is 5 areas of each flat1, flat2, bias1, and bias2 permutation
if (numflat>0&&numbias>0) {
     for (f1=1;f1<=numflat-1;f1+=1){
           for (f2=f1+1;f2<=numflat;f2+=1){
                 for (b1=1;b1<=numbias-1;b1+=1){
                       for (b2=b1+1;b2<=numbias;b2+=1){
                             tabpar ("datalistflatuse",1,f1)
                             flat1=tabpar.value

                             imgets (flat1,param="i_naxis1")
                             xlen=int(imgets.value)
                             imgets (flat1,param="i_naxis2")
                             ylen=int(imgets.value)

                             x1=50
                             x2=150
                             x3=xlen-150
                             x4=xlen-50
                             if ((xlen%2)==0){
                                   x5=(xlen/2)-50
                                   x6=(xlen/2)+50
                             }
                             ;
                             if ((xlen%2)==1){
                                   x5=((xlen-1)/2)-50
                                   x6=((xlen-1)/2)+50
                             }
                             ;

                             y1=50
                             y2=150
                             y3=ylen-150
                             y4=ylen-50
                             if ((ylen%2)==0){
                                   y5=(ylen/2)-50
                                   y6=(ylen/2)+50
                             }
                             ;
                             if ((ylen%2)==1){
                                   y5=((ylen-1)/2)-50
                                   y6=((ylen-1)/2)+50
                             }
                             ;

                             tabpar ("datalistflatuse",1,f2)
                             flat2=tabpar.value
                             tabpar ("datalistbiasuse",1,b1)
                             bias1=tabpar.value
                             tabpar ("datalistbiasuse",1,b2)
                             bias2=tabpar.value

                             clear

                             path

                             print ("")
```

```
                                remaining = remaining - 1
                                print ("Iterations remaining: ",remaining)

                                print ("")

                                print ("Flat 1: ",flat1)
                                print ("Flat 2: ",flat2)
                                print ("Bias 1: ",bias1)
                                print ("Bias 2: ",bias2)

                                ap_findgain
(flat1,flat2,bias1,bias2,section="["//x1//":"//x2//","//y1//":"//y2//"]",verbose-,>>"gainrdnoise.info")
                                ap_findgain
(flat1,flat2,bias1,bias2,section="["//x1//":"//x2//","//y3//":"//y4//"]",verbose-,>>"gainrdnoise.info")
                                ap_findgain
(flat1,flat2,bias1,bias2,section="["//x3//":"//x4//","//y1//":"//y2//"]",verbose-,>>"gainrdnoise.info")
                                ap_findgain
(flat1,flat2,bias1,bias2,section="["//x3//":"//x4//","//y3//":"//y4//"]",verbose-,>>"gainrdnoise.info")
                                ap_findgain
(flat1,flat2,bias1,bias2,section="["//x5//":"//x6//","//y5//":"//y6//"]",verbose-,>>"gainrdnoise.info")
                            }
                    }
                }
            }
}
;

#deletes possibly incompatible datalists
del data*

beep
```

ap_gain_rdnoise_calculator

```
#DOCUMENTATION: ap_gain_rdnoise_calculator_v_2.cl
#    ap_gain_rdnoise_calculator_v_2.cl by Paul Iverson
#
#          ap_gain_rdnoise_calculator_v_2.cl is a script designed to calculate gain and rdnoise data from
gainrdnoise.info files. It is necessary
#          to run the ap_gain_rdnoise_collector_v_3.cl previous to running this script.
#
#          IRAF packages required
#                imutil
#
#          Possible sources of error:
#                Inaccurate filenames  -> no data in datalists

#variable declarations
string     s1=""
string     s2=""
string     gain=""
string     rdnoise=""
struct     *list1
struct     *list2

#deletes possibly incompatible datalists
del data*

#determines average gain and rdnoise
if (access("gainrdnoise.info")){
     list1="gainrdnoise.info"
     while (fscan(list1,s1,s2)!=EOF){
            gain=(s1)
            rdnoise=(s2)

            clear
            print ("Calculating gain and rdnoise")
            if (real(gain)>0){
                  print (gain,>>"gain.info.temp")
            }
            ;
            if (real(rdnoise)>0){
                  print (rdnoise,>>"rdnoise.info.temp")
            }
            ;
     }

     sections ("gain.info.temp")
     if (sections.nimages>0){
            type "gain.info.temp" | average > "gain.info"
     }
     ;
     sections ("rdnoise.info.temp")
     if (sections.nimages>0){
            type "rdnoise.info.temp" | average > "rdnoise.info"
     }
     ;
}
;

#deletes possibly incompatible datalists
del data*
del *temp
```

ap_gain_rdnoise_obhead

```
#DOCUMENTATION: ap_gain_rdnoise_obhead_v_2.cl
#    ap_gain_rdnoise_obhead_v_2.cl by Paul Iverson
#
#           ap_gain_rdnoise_obhead_v_2.cl is a script designed to modify the headers of object and
#           flat frames
#           rfits'ed with the ap_rfits_v_2.cl script.
#
#           This script adds the header fields GAIN and RDNOISE. These fields are utilized later in
#           the ap_align_v_?.cl script.
#           It is necessary to run the ap_gain_rdnoise_collector_v_?.cl and the
#           ap_gain_rdnoise_calculator_v_?.cl
#           scripts on raw bias and flat frames previous to running this script.
#
#           IRAF packages required
#                  imutil
#                  astutil
#
#           Linux commands required
#                  date
#
#           Possible sources of error:
#                  Inaccurate filenames or filter entries in header -> no data in datalists
#                  No .info files from the collector and calculator scripts -> no data in variables

#variable declarations
real Gain=0.0
real RDNoise=0.0
string     s1=""
string     files=""
string     datestring=""
string     timestring=""
struct     *list1

#deletes possibly incompatible datalists
del data*

#extracts filename and subset from rfits'ed object and flat frames
hselect ("*.fits","$I,SUBSET",yes,>>"datalist1")
list1="datalist1"
while (fscan(list1,s1)!=EOF){
     files=(s1)

     #finds gain value from .info file
     if (access("gain.info")){
            tabpar ("gain.info",1,1)
            Gain=real(tabpar.value)
     }
     ;

     #finds rdnoise value from .info file
     if (access("rdnoise.info")){
            tabpar ("rdnoise.info",1,1)
            RDNoise=real(tabpar.value)
     }
     ;

     #edits header fields "GAIN" and "RDNOISE"
     hedit (files,"GAIN",Gain,add+,ver-)
     hedit (files,"RDNOISE",RDNoise,add+,ver-)
}

#deletes possibly incompatible datalists
del data*

#adds date and time of when script was applied to frames to headers
date '+DATE:%m/%d/%y%nTIME:%H:%M:%S' > "datalist"
tabpar ("datalist",1,1)
datestring=tabpar.value
tabpar ("datalist",1,2)
timestring=tabpar.value

hedit ("obj*.fits","AP_GAIN_RDNOISE_OBHEAD",datestring//" "//timestring,add+,ver-)
hedit ("flat*.fits","AP_GAIN_RDNOISE_OBHEAD",datestring//" "//timestring,add+,ver-)

#deletes possibly incompatible datalists
del data*
beep
```

A.1.5 *ap_pierside*

```
#DOCUMENTATION: ap_pierside_obhead_v_1.cl
#    ap_pierside_obhead_v_1.cl by Paul Iverson
#
#          ap_pierside_obhead_v_1.cl is a script designed to modify filenames based on
#          pier side. Since this is an issue only on side mounted telescopes, it only is
#          necessary when reducing frames from such systems. It is made to function on
#          object frames rfits'ed with the ap_rfits_v_2.cl script and header modified
#          by general obhead scripts.
#
#          This script modifies the name of a file based on the side of the pier that is
#          located on. The "E" addition is equivalent to an East-facing orientation whereas
#          the "W" denotes a West-facing orientation. These are primarily determined by
#          calculating the hour angle (LST-RA); however, it has been noted that close to
#          the meridian, a side-mounted telescope can track into positive hour angles
#          without changing orientation. Therefore near HA=0, the script displays the
#          image in ds9 and asks the user to verify whether it is E or W oriented.
#
#          IRAF packages required
#
#
#          Linux commands required
#                date
#
#          Possible sources of error:
#                Inaccurate filess or filter entries in header -> no data in datalists

procedure ap_pierside_obhead_v_1 (East_or_West)

string     East_or_West {prompt="Enter orientation of image",enum="E|W"}

begin
     string EastWest
end

#variable declarations
int  n=1
int  numimg=1
int t=0
int cont=0
real HA=0
real ST=0
real STCheck=0
real RA=0
real exp
real limit
string     s1=""
string     s2=""
string     s3=""
string     s4=""
string     s5=""
string     files=""
string     names=""
struct     *list1


#deletes possibly incompatible datalists
del data*

hselect ("obj*","$I,HJD",yes,>"datalist1")
tsort ("datalist1",2)

sections @datalist1
numimg=sections.nimages

hselect ("obj*","$I",yes,>"datalistfiles")
sed -e 's/.fits//g' "datalistfiles" >"datalistfilenames"

hselect ("obj*","HJD",yes,>"datalistHJD")

hselect ("obj*","ST,RA",yes,>"datalistLSTRA")


joinlines ("datalistfiles,datalistHJD,datalistLSTRA,datalistfilenames",>"datalisttotal")
tsort ("datalisttotal",2)
list1="datalisttotal"
while (fscan(list1,s1,s2,s3,s4,s5)!=EOF){
     files=(s1)
```

```
ST=real(s3)
RA=real(s4)
names=(s5)
if(t==0){
       imgets(files,param="EXPTIME")
       exp = real(imgets.value)
       limit = (exp*10)/60/60
       if(limit<0.33){
              limit=0.33
       }
       ;
       t=1
}
;

if (ST<STCheck){
       ST=ST+24

       HA=ST-RA

       if (HA<-limit){
              hedit (files,"HA",HA,add+,ver-)
              hedit (files,"PIER","E",add+,ver-)
              rename (files,names//"-east",field="root")
              if (n==1){
                     disp (names//"-east.fits",1)
                     n=n+1
              }
              ;
       }
       ;
       if (HA>limit){
              hedit (files,"HA",HA,add+,ver-)
              hedit (files,"PIER","W",add+,ver-)
              rename (files,names//"-west",field="root")

              cont=0
       }
       ;
       if (HA>=-limit&&HA<=limit&&cont!=1){
              if (n!=1){
                     clear
                     print ("Image in frame 1 is in East orientation")
              }
              ;
              if (n==1){
                     disp ("*obj*-"//numimg//"*",1)
                     clear
                     print ("Image in frame 1 is in West orientation")
              }
              ;
              disp (files,2)
              print ("")
              EastWest=East_or_West
              hedit (files,"HA",HA,add+,ver-)
              hedit (files,"PIER",EastWest,add+,ver-)
              if (EastWest == "E"){
                     rename (files,names//"-east",field="root")
              }
              ;
              if (EastWest == "W"){
                     rename (files,names//"-west",field="root")
                     cont = 1
              }
              ;
       }
       ;
       if (cont==1){
              hedit (files,"HA",HA,add+,ver-)
              hedit (files,"PIER","W",add+,ver-)
              rename (files,names//"-west",field="root")
       }
       ;

       STCheck=ST
}
;
```

```
    if (ST>STCheck){
          HA=ST-RA

          if (HA<-limit){
                hedit (files,"HA",HA,add+,ver-)
                hedit (files,"PIER","E",add+,ver-)
                rename (files,names//"-east",field="root")
                if (n==1){
                      disp (names//"-east.fits",1)
                      n=n+1
                }
                ;
          }
          ;
          if (HA>limit){
                hedit (files,"HA",HA,add+,ver-)
                hedit (files,"PIER","W",add+,ver-)
                rename (files,names//"-west",field="root")

                cont=0
          }
          ;
          if (HA>=-limit&&HA<=limit&&cont!=1){
                if (n!=1){
                      clear
                      print ("Image in frame 1 is in East orientation")
                }
                ;
                if (n==1){
                      disp ("*obj*-"//numimg//"*",1)
                      clear
                      print ("Image in frame 1 is in West orientation")
                }
                ;
                disp (files,2)
                print ("")
                EastWest=East_or_West
                hedit (files,"HA",HA,add+,ver-)
                hedit (files,"PIER",EastWest,add+,ver-)
                if (EastWest == "E"){
                      rename (files,names//"-east",field="root")
                }
                ;
                if (EastWest == "W"){
                      rename (files,names//"-west",field="root")
                      cont = 1
                }
                ;
          }
          ;
          if(cont==1){
                hedit (files,"HA",HA,add+,ver-)
                hedit (files,"PIER","W",add+,ver-)
                rename (files,names//"-west",field="root")
          }
          ;

          STCheck=ST
    }
    ;

}

#deletes possibly incompatible datalists
del data*

beep
```

A.1.6 *ap_proc*

```
#DOCUMENTATION: ap_proc_v_3.cl
#    ap_proc_v_3.cl by Paul Iverson
#
#           ap_proc_v_3.cl is a script designed to process bias, dark, flat, and image
#           frames rfits'ed with the ap_rfits_v_2.cl script.
#
#           This script packages the entire reduction process performed previously by
#           hand. Whereas previous versions of this script had interactive modules, this
#           version eliminates interaction in order to increase processing speed. The
#           user is still able to decide which reduction algorithm will be utilized;
#           however, a review of results is no longer supplied within the script. For an
#           interactive version see ap_proc_v_2.cl.
#
#           IRAF packages required
#                  imutil
#                  astutil
#
#           Linux commands required
#                  date
#
#           Possible sources of error:
#                  Inaccurate filenames or filter entries in header -> no data in datalists

procedure ap_proc_v_3 (reduction_alg,flat_alg)

string reduction_alg     {prompt="Reduction algorithm",enum="minmax|ccdclip|avsigclip"}
string flat_alg   {prompt="Flat algorithm",enum="Calibration|Object"}

begin
     string algorithm
     string falgorithm

     algorithm=reduction_alg
     falgorithm=flat_alg
end

#variable declarations
int         n=1
int         m=1
int         numimg
int         high, low, keep
real        highsigma, lowsigma
string      s1=""
string      s2=""
string      s3=""
string      s4=""
string      files=""
string      names=""
string      filter=""
string      checkfilter=""
string      pier=""
string      checkpier=""
string      datestring=""
string      timestring=""
struct      *list1
struct      *list2
struct      *list3

#deletes possibly incompatible datalists and proc files
del data*
del *.proc*
del Flat*
del Zero*
del Dark*
del Illum*

sections ("zero*.fits")
numimg=sections.nimages

#determines high and low number of bias pixels to reject
high=nint(sections.nimages*0.25)
low=nint(sections.nimages*0.25)

highsigma=1.5
lowsigma=1.5

#determines the number of bias pixels to keep
```

```
keep=nint(sections.nimages*0.5)

#combines bias frames
zerocombine
("zero*.fits",reject=algorithm,nlow=low,nhigh=high,nkeep=keep,lsigma=lowsigma,hsigma=highsigma,rdnoise="RDNOISE
",gain="GAIN")

#processes dark frames with combined Zero.fits frame
sections ("dark*.fits",opt="fullname",>"datalistdark")
list1="datalistdark"
while (fscan(list1,s1)!=EOF){
     files=(s1)

     ccdproc (files,output=(files)//".proc",ccdtype="dark",fixpix-,overscan-,trim-,zerocor+,darkcor-,flatcor-
,zero="Zero.fits")
}
del data*

sections ("dark*proc.fits")

#determines high and low number of dark pixels to reject
high=nint(sections.nimages*0.25)
low=nint(sections.nimages*0.25)

#determines the number of dark pixels to keep
keep=nint(sections.nimages*0.5)

#combines dark frames
darkcombine
("dark*proc.fits",reject=algorithm,nlow=low,nhigh=high,nkeep=keep,lsigma=lowsigma,hsigma=highsigma,rdnoise="RDN
OISE",gain="GAIN")

if (falgorithm=="Calibration"){
     #processes flat frames with combined zero-corrected Dark.fits; combines flats by filter
     hselect ("flat*","$I,FILTER",yes,>"datalistflat")
     tsort ("datalistflat",2)
     list1="datalistflat"
     while (fscan(list1,s1,s2)!=EOF){
            filter=(s2)

            if (checkfilter!=filter){
                  print (filter,>>"datalistfilter")

                  sections ("flat"//filter//"*.fits",opt="fullname",>"datalistflat"//filter)
                  list2="datalistflat"//filter
                  while (fscan(list2,s3)!=EOF){
                        files=(s3)

                        ccdproc (files,output=(files)//".proc",ccdtype="flat",fixpix-,overscan-,trim-
,zerocor+,darkcor+,flatcor-,zero="Zero.fits",dark="Dark.fits")
                  }

                  sections ("flat"//filter//"*proc.fits")

                  #determines high and low number of flat pixels to reject
                  high=nint(sections.nimages*0.25)
                  low=nint(sections.nimages*0.25)

                  #determines the number of flat pixels to keep
                  keep=nint(sections.nimages*0.5)

                  flatcombine
("flat"//filter//"*proc.fits",reject=algorithm,subsets+,nlow=low,nhigh=high,nkeep=keep,lsigma=lowsigma,hsigma=h
ighsigma,rdnoise="RDNOISE",gain="GAIN")

                  mkillumcor ("flat"//filter//"*proc.fits","Illum"//filter)
            }
            ;
            checkfilter=filter
     }

     #processes object frames by filter with combined zero-corrected, dark-corrected
     #flat frames
     sections ("obj*.fits",opt="fullname",>"datalistobj")
     list1="datalistobj"
     while (fscan(list1,s1)!=EOF){
            files=(s1)
```

```
           ccdproc (files,output=(files)//".proc",ccdtype="object",fixpix-,overscan-,trim-
,zerocor+,darkcor+,flatcor+,illumco+,zero="Zero.fits",dark="Dark.fits",flat="Flat*.fits",illum="Illum*.fits")
     }
}
;
if (falgorithm=="Object"){
     #processes flat frames with combined zero-corrected Dark.fits; combines flats by filter
     hselect ("obj*","$I,FILTER,PIER",yes,>"datalistobjflat")
     tsort ("datalistobjflat",3)
     list1="datalistobjflat"
     while (fscan(list1,s1,s2,s3)!=EOF){
           pier=(s3)
           if (checkpier!=pier){
                 print (pier,>>"datalistpier")
           }
           ;
           checkpier=pier
     }
     checkpier=""
     pier=""

     tsort ("datalistobjflat",2)
     list1="datalistobjflat"
     while (fscan(list1,s1,s2,s3)!=EOF){
           filter=(s2)
           if (checkfilter!=filter){
                 print (filter,>>"datalistfilter")
           }
           ;
           checkfilter=filter
     }
     checkfilter=""
     filter=""

     list1="datalistpier"
     while (fscan(list1,s1)!=EOF){
           pier=(s1)
           list2="datalistobjflat"
           while (fscan(list2,s2,s3,s4)!=EOF){
                 files=(s2)
                 filter=(s3)
                 if ((s4)==pier){
                       print (files,>>"datalist"//pier//"files")
                       print (filter,>>"datalist"//pier//"filter")
                 }
                 ;
           }
           joinlines ("datalist"//pier//"files,datalist"//pier//"filter",>"datalist"//pier//"total")

           list2="datalist"//pier//"total"
           while (fscan(list2,s2,s3)!=EOF){
                 files=(s2)
                 filter=(s3)
                 if (m==4){
                       m=1
                 }
                 ;
                 if (m==1){
                       copy (s2,"flat"//filter//"_obj-"//pier//"-"//n//".fits")
                       hedit ("flat"//filter//"_obj-"//pier//"-"//n//".fits","IMAGETYP","flat",add+,ver-)
                       hedit ("flat"//filter//"_obj-"//pier//"-"//n//".fits","SUBSET",filter,add+,ver-)
                       hedit ("flat"//filter//"_obj-"//pier//"-"//n//".fits","OBJECT","Flat
"//filter,add+,ver-)
                       n=n+1
                 }
                 ;
                 m=m+1
           }
           n=1
           m=1

           list2="datalistfilter"
           while (fscan(list2,s2)!=EOF){
                 filter=(s2)

                 sections ("flat"//filter//"_obj-"//pier//"*.fits",>"datalistflat"//filter//"_"//pier)
```

```
                  list2="datalistflat"//filter//"_"//pier
                  while (fscan(list2,s3)!=EOF){
                        files=(s3)
                        ccdproc (files,output=(files)//".proc",ccdtype="flat",fixpix-,overscan-,trim-
,zerocor+,darkcor+,flatcor-,zero="Zero.fits",dark="Dark.fits")
                  }

                  sections ("flat"//filter//"*proc.fits")

                  #determines high and low number of flat pixels to reject
                  high=nint(sections.nimages*0.25)
                  low=nint(sections.nimages*0.25)

                  #determines the number of flat pixels to keep
                  keep=nint(sections.nimages*0.5)

                  flatcombine
("flat"//filter//"*proc.fits",output="FlatObj"//filter//"_"//pier,reject=algorithm,subsets+,nlow=low,nhigh=high
,nkeep=keep,lsigma=lowsigma,hsigma=highsigma,rdnoise="RDNOISE",gain="GAIN")

                  mkillumcor ("flat"//filter//"*proc.fits","Illum"//filter//"_"//pier)
            }
      }

      #processes object frames by filter with combined zero-corrected, dark-corrected
      #flat frames
      sections ("obj*.fits",opt="fullname",>"datalistobj")
      list1="datalistobjflat"
      while (fscan(list1,s1,s2,s3)!=EOF){
            files=(s1)
            pier=(s3)

            ccdproc (files,output=(files)//".proc",ccdtype="object",fixpix-,overscan-,trim-
,zerocor+,darkcor+,flatcor+,illumco+,zero="Zero.fits",dark="Dark.fits",flat="FlatObj*"//pier//"*.fits",illum="I
llum*"//pier//".fits")
      }
}
;

#deletes possibly incompatible datalists and log files
del data*
del log*

#creates pre-processed frames folder
if(access("filespreproc")!=yes){
      mkdir filespreproc
}
;
if(access("filespreproc")==yes){
      cd filespreproc
      del *.*
      cd ..
}
;

#moves pre-processed frames to folder "filespreproc"
sections ("*.*",opt="fullname",>"datalistfiles")
match ("proc","datalistfiles",stop+,>>"datalistnonprocA")
match ("info","datalistnonprocA",stop+,>>"datalistnonprocB")
match ("Flat","datalistnonprocB",stop+,>>"datalistnonprocC")
match ("Dark","datalistnonprocC",stop+,>>"datalistnonprocD")
match ("Zero","datalistnonprocD",stop+,>>"datalistnonprocE")
match ("Illum","datalistnonprocE",stop+,>>"datalistnonprocF")
match ("reg","datalistnonprocF",stop+,>>"datalistnonprocfiles")
list1 = "datalistnonprocfiles"
while (fscan(list1,s1)!=EOF){
      files=(s1)

      move (files,"filespreproc")
}

#deletes possibly incompatible datalists
del data*

sections ("*.proc*",>"datalist1")
sed -e 's/.fits//g' "datalist1" >"datalist2"
joinlines ("datalist1,datalist2",>"datalistfilenames")
```

```
list1="datalistfilenames"
while (fscan(list1,s1,s2)!=EOF){
     files=(s1)
     name=(s2)
     rename (files,name//".fits",field="all")
}

#deletes possibly incompatible datalists
del data*

#adds date and time of when script was applied to frames to headers
date '+DATE:%m/%d/%y%nTIME:%H:%M:%S' > "datalist"
tabpar ("datalist",1,1)
datestring=tabpar.value
tabpar ("datalist",1,2)
timestring=tabpar.value

hedit ("*.fits","AP_PROC",datestring//" "//timestring,add+,ver-)
hedit ("*.fits","PROC_ALG",algorithm,add+,ver-)

#deletes possibly incompatible datalists
del data*
del subsets

beep
```

A.1.7 *ap_skynoise*

```
#DOCUMENTATION: ap_skynoise_obhead_v_2.cl
#    ap_skynoise_obhead_v_2.cl by Paul Iverson
#
#          ap_skynoise_obhead_v_2.cl is a script designed to modify the headers of object frames rfits'ed with
#          the ap_rfits_v_2.cl script.
#
#          This script adds the header fields SKYAVG and SKYDEV.
#
#          IRAF packages required
#          imutil
#                astutil
#
#          Linux commands required
#                date
#
#          Possible sources of error:
#                Inaccurate filenames or filter entries in header -> no data in datalists

#variable declarations
int         x1=0
int         x2=0
int         y1=0
int         y2=0
int         xlen=0
int         ylen=0
int         n=3
real        skyvalue=0.0
real        skydev=0.0
string      s1=""
string      files=""
string      datestring=""
string      timestring=""
struct      *list1

#deletes possibly incompatible datalists
del data*
del *trim*
del *mag*
del skydao.reg

sections ("*obj*.fits",opt="fullname",>"datalist1")
list1="datalist1"
while (fscan(list1,s1)!=EOF){
     files = (s1)

     if (access("sky.reg")==yes){
           del sky.reg
     }
     ;

     imgets (files,param="i_naxis1")
     xlen=int(imgets.value)
     imgets (files,param="i_naxis2")
     ylen=int(imgets.value)

     x1=10
     x2=xlen-10
     y1=10
     y2=ylen-10

     #copies smaller field of frame to remove possible overscan areas
     imcopy (files//"["//x1//":"//x2//","//y1//":"//y2//"]",files//".trim")

     findpars.threshold=10
     datapars.sigma=20
     datapars.datamin=INDEF
     datapars.datamax=INDEF
     fitskypars.annulus=10
     fitskypars.dannulu=4

     findpars.threshold.p_mode = "h"
     datapars.fwhmpsf.p_mode = "h"
     datapars.sigma.p_mode = "h"
     datapars.datamin.p_mode = "h"
     datapars.datamax.p_mode = "h"
     phot.output.p_mode = "h"
```

A.1.8 *ap_fwhm_obhead*

```
#DOCUMENTATION: ap_fwhm_obhead_v_4.cl
#    ap_fwhm_obhead_v_4.cl by Paul Iverson
#
#           ap_fwhm_obhead_v_4.cl is a script designed to modify the headers of object
#           frames rfits'ed with the ap_rfits_v_2.cl script.
#
#           This script adds the header field FWHM.
#
#           IRAF packages required
#                  imutil
#                  astutil
#
#           Linux commands required
#                  date
#
#           Possible sources of error:
#                  Inaccurate filenames or filter entries in header -> no data in datalists

#variable declarations
int         i
int         n=1
int         m=1
int         xlen
int         ylen
int         x1,x2
int         y1,y2
real        num
real        avg,dev
real per=1
real        standdev
real skydev
real flux
real fwhmxA=0
real fwhmxB=0
real fwhmyA=0
real fwhmyB=0
real temp3
string      s1=""
string      s2=""
string      s3=""
string      s4=""
string      s5=""
string      files,trimfiles
string      temp
string      temp2
string      temp4
string      datestring=""
string      timestring=""
struct      *list1
struct      *list2

#deletes possibly incompatible datalists and other files
del data*
del fwhm*
del nomo*
del *trim*

#adds date and time of when script was applied to frames to headers
date '+DATE:%m/%d/%y%nTIME:%H:%M:%S' > "datalist"
tabpar ("datalist",1,1)
datestring=tabpar.value
tabpar ("datalist",1,2)
timestring=tabpar.value

hedit ("obj*.fits","AP_FWHM_OBHEAD",datestring//" "//timestring,add+,ver-)

#deletes possibly incompatible datalists
del data*

hselect ("*obj*.fits","$I,SKYDEV",yes,>"datalistfilenames")
list1 = "datalistfilenames"
while (fscan(list1,s1,s2)!=EOF){
     files=(s1)
     skydev=real(s2)

     imgets (files,param="i_naxis1")
     xlen=int(imgets.value)
```

```
imgets (files,param="i_naxis2")
ylen=int(imgets.value)

x1=10
x2=xlen-10
y1=10
y2=ylen-10

#copies smaller field of frame to remove possible overscan areas
imcopy ((files)//"["//x1//":"//x2//","//y1//":"//y2//"]",(files)//".trim")

trimfiles=(files)//".trim.fits"

if (access("fwhm.reg")==yes){
      del fwhm.reg
}
;

findpars.threshold=2*skydev

centerpars.calg="centroid"
centerpars.cbox=10
#cthreshold setting from Hintz's newton
centerpars.cthreshold=0.75
centerpars.cmaxiter=20
centerpars.maxshift=8.5

fitskypars.salgorithm="mode"
fitskypars.annulus=16
fitskypars.dannulu=4
#sloclip and shiclip setting from Hintz's newton
fitskypars.sloclip=0.
fitskypars.shiclip=10.

photpars.weighting="constant"
photpars.apertures=12
photpars.zmag=30

datapars.fwhmpsf=4
datapars.sigma=skydev
datapars.datamin=INDEF
datapars.datamax=INDEF
datapars.ccdread="RDNOISE"
datapars.gain="GAIN"
datapars.exposure="EXPTIME"
datapars.airmass="AIRMASS"
datapars.filter="SUBSET"
datapars.obstime="HJD"

psfmeasure.radius=5
psfmeasure.sbuffer=5
psfmeasure.swidth=5

#creates coordinate file for fwhm determination
daofind (trimfiles,output="fwhmphot.reg",verif-)
phot (trimfiles,coords="fwhmphot.reg",interac-,verif-)

del fwhmphot.reg

txdump (trimfiles//".mag.1","flux,xcenter,ycenter",yes,>"datalistphot")
list2="datalistphot"
while (fscan(list2,s3,s4,s5)!=EOF){
      flux=real(s3)
      if (flux>15000.0&&m<=50){
            print (s4,>>"datalistx")
            print (s5,>>"datalisty")
            m=m+1
      }
      ;
}
m=1
del "datalistphot"

joinlines ("datalistx,datalisty",>"datalistphot")
del "datalistx"
del "datalisty"
```

```
	tsort ("datalistphot","1")
	list2="datalistphot"
	while (fscan(list2,s3,s4)!=EOF){
		fwhmxA=real(s3)
		fwhmyA=real(s4)

		if (fwhmxA>fwhmxB+10){
			print (fwhmxA,>>"datalistx")
			print (fwhmyA,>>"datalisty")
		}
		;
		if (fwhmxA<fwhmxB+10){
			if (fwhmyA>fwhmyB+10||fwhmyA<fwhmyB-10){
				print (fwhmxA,>>"datalistx")
				print (fwhmyA,>>"datalisty")
			}
			;
		}
		;

		fwhmxB=fwhmxA
		fwhmyB=fwhmyA
	}

	temp4=access("datalistx")
	if (temp4=="no"){
		del (files//"*")
		del "datalistphot"
	}
	;
	if (temp4!="no"){
		joinlines ("datalistx,datalisty",>"fwhm.reg")

		del "datalistx"
		del "datalisty"
		del "datalistphot"

		sections @fwhm.reg
		if (sections.nimages>0){
			print ("q",>>"nomoreqs")

			psfmeasure.radius=10
			psfmeasure.iterati=5

			#determines psf fwhm of coordniate file objects
			psfmeasure (trimfiles,display-,imagecur="fwhm.reg",graphcur="nomoreqs",>(trimfiles)//".fwhm")

			del "nomoreqs"

			copy ((trimfiles)//".fwhm","datafwhmA")
			sed -e '/NOAO/d' -e '/Image/d' -e '/Average/d' -e 's/" "/d/g' -e 's/obj-
............/ddddddddddddddd/g' "datafwhmA" >> "datafwhmB"

			sections @datafwhmB
			clear

			list2 = "datafwhmB"
			while (fscan(list2,s3) != EOF){
				temp=(s3)
				i=strlen(temp)
				if (n!=1&&n<=sections.nimages+1){
					temp=substr(temp,41,i-15)
					print(temp,>>"datafwhmC")
				}
				;
				n=n+1
			}
			n=1

			sed -e 's/d//g' "datafwhmC" >> "datafwhmD"

			#calculates fwhm average and standard deviation
			type "datafwhmD" | average > "datafwhmE"

			tabpar ("datafwhmE",1,1)
			avg=real(tabpar.value)
			tabpar ("datafwhmE",2,1)
```

```
                standdev=real(tabpar.value)

                #removes outlying data points until minimum standdev of 0.477*avg; this represents 95.449% of
data within 2 sigma
                while (standdev>=0.477*avg){
                        list2="datafwhmD"
                        while (fscan(list2,s4)!=EOF){
                                num=real(s4)
                                dev=per*avg
                                if (num>=avg&&num<=avg+dev){
                                        print (num,>>"datafwhmF")
                                }
                                ;
                                if (num<=avg&&num>=avg-dev){
                                        print (num,>>"datafwhmF")
                                }
                                ;
                        }

                        del "datafwhmD"
                        del "datafwhmE"

                        type "datafwhmF" | average > "datafwhmE"

                        tabpar ("datafwhmE",1,1)
                        avg=real(tabpar.value)
                        tabpar ("datafwhmE",2,1)
                        standdev=real(tabpar.value)

                        rename ("datafwhmF","datafwhmD")

                        sections @datafwhmD
                        if (sections.nimages<=4){
                                break
                        }
                        ;
                        per=per-0.01
                }
                per=1

                #cuts fwhm average values to less than 6 digits
                tabpar ("datafwhmE",1,1)
                temp=tabpar.value
                i=strlen(temp)
                if (i>6){
                        avg=real(substr(temp,1,6))
                }
                ;
                if (i<=6){
                        avg=real(temp)
                }
                ;

                #edits header field "FWHM"
                hedit (files,"FWHM",avg,add+,ver-)

                #deletes possibly incompatible datalists
                del datafwhm*
        }
        ;

        #deletes file if unable to determine fwhm
        if (sections.nimages==0){
                del (files)
                del (trimfiles//"*")
        }
        ;
    }
    ;
}

#deletes possibly incompatible datalists
del *mag*
del *trim*
del fwhm*

#extracts filename and fwhm value from object frames
```

```
hselect ("obj*proc*.fits","$I,FWHM",yes,>>"datalist1")

fields ("datalist1",1,>"datalistnames")
fields ("datalist1",2,>"datalistFWHM")

#calculates non-INDEF-included fwhm
match ("INDEF","datalistFWHM",stop+,>>"datalistnoINDEFFWHM")
type "datalistnoINDEFFWHM" | average > "datalistaverageFWHM"
tabpar ("datalistaverageFWHM",1,1)
temp2=tabpar.value
i=int(strlen(temp2))
if (i>6){
     avg=real(substr(temp2,1,6))
}
;
if (i<=6){
     avg=real(temp2)
}
;

#removes INDEFs from fwhm values and replaces them with group average
list1="datalistFWHM"
while (fscan(list1,s1)!=EOF){
     temp2=(s1)
     temp3=5*avg

     if (temp2!="INDEF"){
            if (real(temp2)>temp3){
                  temp2=avg
            }
            ;
            print (temp2,>>"datalistcorrected")
     }
     ;
     if (temp2=="INDEF"){
            temp2=avg
            print (temp2,>>"datalistcorrected")
     }
     ;
}

joinlines ("datalistnames,datalistcorrected",>> "datalisttotal")

list1="datalisttotal"
while (fscan(list1,s1,s2) !=EOF){
     files=(s1)
     avg=real(s2)

     #edits header field "FWHM"
     hedit (files,"FWHM",avg,add+,ver-)
}

#deletes possibly incompatible datalists
del data*


beep
```

A.1.9 *ap_rotate*

```
#DOCUMENTATION: ap_rotate_v_1.cl
#    ap_rotate_v_1.cl by Paul Iverson
#
#          ap_rotate_v_1.cl is a script designed to rotate object frames rfits'ed with
#          the ap_rfits_v_2.cl script.
#
#          This script rotates frames a user defined degree.
#
#          IRAF packages required
#                imutil
#                astutil
#
#          Linux commands required
#                date
#
#          Possible sources of error:
#                Inaccurate filenames or filter entries in header -> no data in datalists

procedure ap_rotate_v_1 (object_name,rotation_degree)

string object_name       {prompt="Enter search parameters for the files you wish to rotate (i.e. star*B)"}
string rotation_degree   {prompt="Enter the degree of desire rotation"}

begin
     string search
     string rot
     search=object_name
     rot=rotation_degree
end

#variable declarations
string      s1=""
string      files=""
string      datestring=""
string      timestring=""
struct      *list1

#deletes possibly incompatible datalists
del data*

#adds date and time of when script was applied to frames to headers
date '+DATE:%m/%d/%y%nTIME:%H:%M:%S' > "datalist"
tabpar ("datalist",1,1)
datestring=tabpar.value
tabpar ("datalist",1,2)
timestring=tabpar.value

sections ((search)//"*",>"datalistrotate")
list1="datalistrotate"
while (fscan(list1,s1) != EOF){
     files=(s1)
     rotate (files,files,int(rot))

     hedit (files,"AP_ROTATE",datestring//" "//timestring,add+,ver-)
     hedit (files,"ROTATION_DEGREE",rot,,add+,ver-)
}

#deletes possibly incompatible datalists
del data*

beep
```

A.1.10 *ap_trim*

```
#DOCUMENTATION: ap_trim_v_2.cl
#    ap_trim_v_2.cl by Paul Iverson
#
#          ap_trim_v_2.cl is a script designed to trim object frames rfits'ed with
#          the ap_rfits_v_2.cl script.
#
#          This script trims possible overscan areas from frames. Default trim value is 10 pixels.
#
#          IRAF packages required
#                imutil
#                astutil
#
#          Linux commands required
#                date
#
#          Possible sources of error:
#                Inaccurate filenames or filter entries in header -> no data in datalists

#variable declarations
int         n = 1
int         xlen
int         ylen
int         x1,x2
int         y1,y2
string      s1=""
string      files=""
string      datestring=""
string      timestring=""
struct      *list1

#deletes possibly incompatible datalists
del data*

#copies smaller field of frame to remove possible overscan areas
sections ("*obj*.fits",opt="fullname",>"datalistfile")
sed -e 's/.fits//g' "datalistfile" >"datalistfilenames"
list1="datalistfilenames"
while (fscan(list1,s1)!=EOF){
     files=(s1)

     imgets (files,param="i_naxis1")
     xlen=int(imgets.value)
     imgets (files,param="i_naxis2")
     ylen=int(imgets.value)

     x1=10
     x2=xlen-10
     y1=10
     y2=ylen-10

     imcopy ((files)//"["//x1//":"//x2//","//y1//":"//y2//"]",(files)//"-trim")
}

#moves non-trimmed files into filescomp folder
sections ("*.*",opt="fullname",>>"datalist1")
match ("trim","datalist1",stop+,>>"datalist2")
match ("info","datalist2",stop+,>>"datalist3")
match ("reg","datalist3",stop+,>>"datalist4")
list1 = "datalist4"
while (fscan(list1,s1) != EOF){
     if(access("filescomp") && n == 1){
           cd filescomp
           del *
           cd ..
           movefiles (s1,"filescomp")
     }
     ;

     if(access("filescomp") && n != 1){
           movefiles (s1,"filescomp")
     }
     ;

     if(access("filescomp")!=yes){
           mkdir filescomp
           movefiles (s1,"filescomp")
     }
```

```
    ;
    n=n+1
}

#deletes possibly incompatible datalists
del data*

#adds date and time of when script was applied to frames to headers
date '+DATE:%m/%d/%y%nTIME:%H:%M:%S' > "datalist"
tabpar ("datalist",1,1)
datestring=tabpar.value
tabpar ("datalist",1,2)
timestring=tabpar.value

hedit ("obj*.fits","AP_TRIM",datestring//" "//timestring,add+,ver-)

#deletes possibly incompatible datalists
del data*

beep
```

A.1.11 *ap_align*

```
#ap_align.cl by Paul Iverson
#simplifies call to Eran Ofek's autoalign.cl

procedure ap_align_v_1 (align)

string align      {prompt = "Enter search parameters for the files to align (i.e. star*B)"}

begin
     string alignment
     alignment=align
end

#variable declarations
string s1
string s2
string s3

del data*

#autoalign ("ilist", "prefix", fwhm, readnoise, gain, xytol, objectn, "bug_log")
sections ((alignment)//"*")
if (sections.nimages>0){
     hselect ((alignment)//"*","$I,FWHM,RDNOISE,GAIN,HJD",yes, > "datalist1")
}
;

fields ("datalist1",1,>"datalistfile")
fields ("datalist1",5,>"datalistHJD")

joinlines ("datalistfile,datalistHJD",>>"datalistfilesort")
tsort ("datalistfilesort",2)

fields ("datalistfilesort",1,>"datalistfiles")

fields ("datalist1",2,>"datalist2a")
fields ("datalist1",3,>"datalist3a")
fields ("datalist1",4,>"datalist4a")
type "datalist2a" | average >> "datalist2b"
type "datalist3a" | average >> "datalist3b"
type "datalist4a" | average >> "datalist4b"
fields ("datalist2b",1,>"datalistFWHM")
fields ("datalist3b",1,>"datalistRDNOISE")
fields ("datalist4b",1,>"datalistGAIN")

joinlines ("datalistFWHM,datalistRDNOISE,datalistGAIN", >>"datatotal")

list = "datatotal"
while (fscan(list,s1,s2,s3)!=EOF){
     autoalign ("datalistfiles","a-",real(s1),real(s2),real(s3),1.5,50,"bug_log")
}

del data*
del input_shift
del tmp*
del obj*.fits.*afnl*
del obj*.fits.*coo*
del obj*.fits.*smag*
del obj*.fits.*rmag*
del bug_log

beep
```

A.1.12 *autoalign*

```
procedure autoalign (ilist, prefix, fwhm, readnoise, gain, xytol, objectn, bug_log, succeed)
#-----------------------------------------------------------------------
# autoalign.cl  -
#
#   Documentation
#   -------------
#
# the list:
#   #field name, and list of images in the following lines, etc.
# Example:
#    #GRB990316
#    990316.015
#    990316.016
#    990316.017
#    #AD Leo
#    990316.018
#    990316.019
#    .
#    .
#    .
#
# INSTALL: edit and add the following lines to the login.cl
#        task    $xyshift = $/home/wise-cdr/eran/iraf/bin/xyshift
#        task autodaofind   = /home/wise-cdr/eran/iraf/script/autodaofind.cl
#
#
#
# Written By Eran Ofek, October 1998, Last update: 061098
#-----------------------------------------------------------------------

string  ilist     {"",prompt="list of images to align"}
string  prefix    {"a",prompt="prefix for shifted output images"}
real    fwhm      {1.5,prompt="PSF FWHM in pixels"}
real    readnoise {29.0609,prompt="CCD read out noise in electrons"}
real    gain      {4.0364,prompt="CCD gain in electrons per count"}
real    xytol     {1.5, min=0.0,prompt="matching tolerance for pgshift"}
int     objectn   {50, prompt="Max. Number of stars to match"}
string  bug_log   {"buglog",prompt="logfile name"}
bool    succeed   {no,prompt="succeeded to find astrometric solution"}
real shiftx
real shifty

struct *lis1
struct *lis2
struct *lis3
struct *lis4

begin

string   imname
string   refimage
string   magfile
real     avshiftx     # shift in X axis.
real     avshifty     # shift in Y axis.
real     pershift     # number of stars used to shift the image.
bool     last_ast
int      match_n      # number of stars matched
string images
images = ilist

end

delete (bug_log,verify=no,>>&"/dev/null")

#-----------------------------------
#create list without '#'
#-----------------------------------
#delete ('tmp_ilist',verify=no,>>&"/dev/null")
#lis2 = ilist
#while (fscan(lis2, imname)!=EOF)
#{
#   if (substr(imname,1,1) == '#')
#   {
#      #jump to next line
#   }
#   else
#   {
```

```
#     print (imname, >> 'tmp_ilist')
#   }
#}


# call autodaofind
#lis1 = 'tmp_ilist'
lis1 = ilist
while (fscan(lis1, imname)!= EOF)
{
   autodaofind(imname=imname,out_file="default",fwhm=fwhm,readnoise=readnoise,gain=gain,threshold_sig=10)
}


# find shifts between images
lis3 = ilist
last_ast = yes
while (fscan(lis3, imname)!=EOF)
{
   print ('----------------------------------')
   print (' Field Line : ',imname)
   print ('----------------------------------')

   if (substr(imname,1,1) == '#')
   {
      # next field
      last_ast = yes
   }
   else
   {
      if (last_ast==yes)
      {
         last_ast = no
         # set image to be reference image
         refimage = imname
         print ('==================================')
         print ('Reference Image : ', refimage)
         print ('==================================')
         magfile = imname // '.coo.1'
         #---------------------------------------
         # Prepare the image list file for pgshift
         #---------------------------------------
         print("Prepare catalog for findshift")
         delete ('tmp_object',verify=no,>>&"/dev/null")

         print(imname, > 'tmp_object')

         #creating the .smag file
         delete (imname//'.smag.1',verify=no,>>&"/dev/null")
         txdump(textfile=magfile,fields="ID,XCENTER,YCENTER,MAG,MERR,MSKY,NITER,SHARPNESS,CHI",expr="MAG[1]
!=INDEF",headers=yes,
>>imname//'.smag.1')

         delete (imname//'.rmag.1',verify=no,>>&"/dev/null")
         txdump(textfile=magfile,fields="ID,XCENTER,YCENTER,MAG,MERR,MSKY,NITER,SHARPNESS,CHI",expr="MAG[1]
!=INDEF",headers=no, >>imname//'.rmag.1')

         print(' sorting '//imname//'.rmag.1')
         delete (imname//'.afnl.1',verify=no,>>&"/dev/null")

         # creating the .afnl file
         # sort the rls file by decreasing magnitude
         sort(input_fi=imname//'.rmag.1',column=4,numeric=yes, >> imname//'.afnl.1')

         imcopy (input=imname, output=prefix//imname)

         shiftx = 0
         shifty = 0

         hedit (prefix//imname,"X_PIXEL_SHIFT",shiftx)
         hedit (prefix//imname,"Y_PIXEL_SHIFT",shifty)
      }
      else
      {

         magfile = imname // '.coo.1'
```

```
        #----------------------------------------
        # Prepare the image catalog file for pgshift
        #----------------------------------------
        print("Prepare catalog for findshift")
        delete ('tmp_object',verify=no,>>&"/dev/null")

        print(imname, > 'tmp_object')

        #creating the .smag file
        delete (imname//'.smag.1',verify=no,>>&"/dev/null")
        txdump(textfile=magfile,fields="ID,XCENTER,YCENTER,MAG,MERR,MSKY,NITER,SHARPNESS,CHI",expr="MAG[1]
!=INDEF",headers=yes,
>imname//'.smag.1')

        delete (imname//'.rmag.1',verify=no,>>&"/dev/null")
        txdump(textfile=magfile,fields="ID,XCENTER,YCENTER,MAG,MERR,MSKY,NITER,SHARPNESS,CHI",expr="MAG[1]
!=INDEF",headers=no, >>imname//'.rmag.1')

        print(' sorting '//imname//'.rmag.1')
        delete (imname//'.afnl.1',verify=no,>>&"/dev/null")

        # creating the .afnl file
        # sort the rls file by decreasing magnitude
        sort(input_fi=imname//'.rmag.1',column=4,numeric=yes, >> imname//'.afnl.1')

        #--------------
        # compute shifts
        #--------------
        print (' Computing Shifts for image : ',imname)
        delete ('input_shift',verify=no,>>&"/dev/null")
        print (imname//'.afnl.1', >> 'input_shift')
        print (refimage//'.afnl.1', >> 'input_shift')
        print (xytol, >> 'input_shift')

        print(imname//'.afnl.1','\n',refimage//'.afnl.1','\n',xytol,'\n',objectn) | xyshift | scan(avshiftx,
avshifty, pershift, match_n)

           print("------------- Shift in pixels ----------------")
           print("          X = ", avshiftx)
           print("          Y = ", avshifty)

           print("    % = ", pershift)
           print("    n = ", match_n)
           print('Image : ',imname,'     shift% ',pershift, >> bug_log)


        # shift the image
        print ('   ===> Shifting image : ', imname)
        imshift(input=imname, output=prefix//imname, xshift=-avshiftx, yshift=-avshifty)

        shiftx = 0 - avshiftx
        shifty = 0 - avshifty

        hedit (prefix//imname,"X_PIXEL_SHIFT",shiftx)
        hedit (prefix//imname,"Y_PIXEL_SHIFT",shifty)
     }
   }
}
succeed = yes


print("End...  Bye.")
beep
beep
```

A.1.13 *autodaofind*

```
procedure autodaofind
#-----------------------------------------------------------------
# autodaofind.cl  - Automatic daofind for image.
#
#
#
#
# By : Eran O. Ofek
# Written: August 1998,    Last Update: Aug 10th, 1998
#-----------------------------------------------------------------

string  imname        {"",prompt="image name"}
string  out_file      {"default",prompt="output file name"}
real    fwhm          {3.0,prompt="PSF FWHM in pixels"}
#real    readnoise     {6.50,prompt="CCD read out noise in electrons"}
#real    gain          {8.42,prompt="CCD gain in electrons per count"}
#real    threshold_sig {4.0,prompt="threshold on sigma above background"}
real    readnoise     {30,prompt="CCD read out noise in electrons"}
real    gain          {5,prompt="CCD gain in electrons per count"}
real    threshold_sig {10,prompt="threshold on sigma above background"}

struct *lis1

begin


real       sky_noise
real       im_sigma

hselect ((imname),"SKYDEV",yes,>>"datalistSKYDEV")
tabpar ("datalistSKYDEV",1,1)
sky_noise = real(tabpar.value)

#findthresh (images=imname, gain=gain, readnoi=readnoise, ,coaddtype="average", nframes=1, center="mode",
verbose=no) | scan(sky_noise, im_sigma)


print ('  Find stars using Daofind')

delete (imname//'.coo.*',verify=no,>>&"/dev/null")

daofind(image=imname,output=out_file,verify=no, verbose=yes, sigma=sky_noise, scale=1, fwhmpsf=fwhm,
readnoi=readnoise, epadu=gain, thresho=threshold_sig)

print("END autodaofind")

end
```

A.1.14 *xyshift.f*

```
c  this program calculate the shift between two coordinate-files
c written By Uri Giveon
c modified Eran Ofek

      program xyshift
c23456789 123456789 123456789 123456789 123456789 123456789 123456789
      IMPLICIT NONE

      integer      MaxN
      parameter (MaxN=250)

      integer id,i,j,k,l,m,n
      character*70 ImName
      character*70 ImRef

      integer      ObjectN
      real         x1(MaxN)
      real         y1(MaxN)
      real         x2(MaxN)
      real         y2(MaxN)
      real         shiftx
      real         deltax(MaxN,MaxN)
      real         deltay(MaxN,MaxN)
      real         shifty
      real         num1
      real         num(MaxN,MaxN)
      real         object1
      real         object2
      real         sumx(MaxN,MaxN)
      real         sumy(MaxN,MaxN)
      real         sumx1
      real         sumy1
      real         avshiftx
      real         avshifty
      real         pershift
      real         Tolerance


c------ read data from line
      read (5,'(a70)') ImName
      read (5,'(a70)') ImRef
      read (5,*) Tolerance
      read (5,*) ObjectN

      open(2, file=ImName, status='old')
      open(3, file=ImRef, status='old')

      object1=1.
      object2=1.
50    continue
         read(2,*,end=100)id,x1(object1),y1(object1)
            if(object1.eq.ObjectN)then
               goto 110
            endif
         object1=object1+1
      goto 50
100   object1=object1-1
110   continue
         read(3,*,end=200)id,x2(object2),y2(object2)
            if(object2.eq.ObjectN)then
               goto 210
            endif
         object2=object2+1
      goto 110
200   object2=object2-1


210   close (2)
      close (3)

c------ calculate all the possible differences
      do i=1,object1
         do j=1,object2
            deltax(i,j)=x1(i)-x2(j)
            deltay(i,j)=y1(i)-y2(j)
         enddo
      enddo
```

```
c------ find the shift
       num1=1.
       do k=1,object1
       do l=1,object2
          num(k,l)=1.
          sumx(k,l)=deltax(k,l)
          sumy(k,l)=deltay(k,l)
          do m=k,object1
             if(m.eq.k)then
                goto 600
             endif
          do n=1,object2
             if(deltax(m,n).le.deltax(k,l)+Tolerance.and.deltax(m,n)
     >            .ge.deltax(k,l)-Tolerance)then
                if(deltay(m,n).le.deltay(k,l)+Tolerance.and.deltay(m,n)
     >               .ge.deltay(k,l)-Tolerance)then
                   num(k,l)=num(k,l)+1
                      sumx(k,l)=sumx(k,l)+deltax(m,n)
                      sumy(k,l)=sumy(k,l)+deltay(m,n)
                   goto 600
                endif
             endif
          enddo
600       enddo
             if(num(k,l).gt.num1)then
                num1=num(k,l)
                   sumx1=sumx(k,l)
                      sumy1=sumy(k,l)
                   shiftx=deltax(k,l)
                shifty=deltay(k,l)
             endif
       enddo
       enddo

c------ compute the average shifts & the star percent contributing to it

       avshiftx=sumx1/num1
       avshifty=sumy1/num1
       pershift=100*num1/object1

c------ output data

       write(6,'(f8.2,3x,f8.2,5x,f8.2,5x,f6.1)')
     >          avshiftx,avshifty,pershift,num1

       end
```

A.1.15 *ap_apphot*

```
#DOCUMENTATION:ap_apphot_v_4.cl
#    ap_apphot_v_4.clbyPaulIverson

procedure ap_apphot_v_4(search_align_phot,coordinate_file,phot_method,pier_side_method,flux_per,fwhm_mult)

string search_align_phot {prompt="Enter search parameters for the aligned frames to phot (i.e. a-*star*B)"}
string coordinate_file   {prompt="Enter the name of the coordinatefile (i.e. ds9.reg)"}
string phot_method       {prompt="Enter the phot method to be used",enum="Flux|FWHM"}
string flux_per          {prompt="Enter the percentage of flux to use for each star",enum="80|85|90|95|98"}
string fwhm_mult         {prompt="Enter the multiple of fwhm to use for each star",enum="1.0|1.5|2.0|2.5|3.0"}
string pier_side_method  {prompt="Phot frames by pier side",enum="Yes|No"}

begin
     string searchalignphotometry
     string coordinate
     string phot_meth
     string flux_percent
     string fwhm_multiple
     string pier_side
end

#variable declarations
int         i=1
int         n=1
int         m=1
int         l=1
int         xlen,ylen
int         cont=0
real        skypix
real        xloc,yloc
real        dann=1
real        xshift,yshift
real        xstar,ystar
string      s1=""
string      s2=""
string      s3=""
string      s4=""
string      s5=""
string      s6=""
string      s7=""
string      s8=""
string      s9=""
string      s10=""
string      files
string      filter
string      checkfilter=""
string      inttime=""
string      checkinttime=""
string      lstfiles=""
string      star=""
string      checkstar=""
string      magnitude=""
string      HJD=""
string      airmass=""
string      pier=""
string      checkpier=""
string      datestring=""
string      timestring=""
string      X=""
string      Y=""
string      temp
struct      *list1
struct      *list2
struct      *list3

#deletes possibly incompatible datalists
delete ("data*")
delete ("onframe*")
delete ("nomo*")
delete ("radius.*")
delete ("fwhm.*")
delete ("*temp*")
delete ("star-*")

searchalignphotometry=search_align_phot
coordinate=coordinate_file
phot_meth=phot_method
```

```
if (phot_meth=="Flux"){
     flux_percent=flux_per
}
;
if (phot_meth=="FWHM"){
     fwhm_multiple=fwhm_mult
}
;

pier_side=pier_side_method

sections ((searchalignphotometry)//"*.fits",opt="fullname",>>"datalistfile")
if (sections.nimages>0){
     hselect
((searchalignphotometry)//"*.fits","$I,FWHM,GAIN,SKYDEV,HJD,RDNOISE,X_PIXEL_,Y_PIXEL_,PIER,SUBSET",yes,>>"datat
otal")
     tsort ("datatotal",5)
}
;

hselect ((searchalignphotometry)//"*.fits","PIER",yes,>>"datapier")
tsort ("datapier",1)
list2="datapier"
while (fscan(list2,s1)!=EOF){
     pier=(s1)
     if (pier!=checkpier){
           print (pier,>>"datalistpier")
     }
     ;
     checkpier=pier
}

#datalist contains files, fwhmpsf, epadu, sigma, HJD, rdnoise, pixel shifts, pier, subset
sections ("@datatotal")
if (sections.nimages>0){
     list1="datatotal"
     while (fscan(list1,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10)!=EOF){
           clear
           print ("File:",s1)
           print ("HJD:",s5)
           print ("")
           print ("Subset:",s10)
           print ("")
           print ("Pier:",s9)
           print ("")
           print ("FWHM:",s2)
           print ("")
           print ("Gain:",s3)
           print ("RDNoise:",s6)
           print ("Sky Deviation:",s4)
           print ("")
           print ("X Pixel Shift:",s7)
           print ("Y Pixel Shift:",s8)

           sleep (1)

           centerpars.calg="centroid"
           centerpars.cbox=3*real(s2)
           #cthresholdsettingfromHintz'snewton
           centerpars.cthreshold=0.75
           centerpars.cmaxiter=20
           centerpars.maxshift=8.5

           fitskypars.salgorithm="mode"
           fitskypars.annulus=10*real(s2)
           fitskypars.dannulu=dann
           #sloclipandshiclipsettingfromHintz'snewton
           fitskypars.sloclip=0.
           fitskypars.shiclip=10.

           photpars.weighting="constant"
           photpars.apertur=3*real(s2)
           photpars.zmag=30

           datapars.fwhmpsf=real(s2)
           datapars.sigma=real(s4)
```

```
datapars.datamin=INDEF
datapars.datamax=INDEF
datapars.ccdread="RDNOISE"
datapars.gain="GAIN"
datapars.readnoise=real(s6)
datapars.epadu=real(s3)
datapars.exposure="EXPTIME"
datapars.airmass="AIRMASS"
datapars.filter="SUBSET"
datapars.obstime="HJD"

psfmeasure.radius=5
psfmeasure.sbuffer=5
psfmeasure.swidth=5

files=(s1)
xshift=real(s7)
yshift=real(s8)

pier=(s9)

if (m==1){
       imgets (files,param="i_naxis1")
       xlen=int (imgets.value)
       imgets (files,param="i_naxis2")
       ylen=int (imgets.value)


       list2=(coordinate)
       while (fscan(list2,X,Y)!=EOF){
              print (X,>"datalistx")
              print (Y,>"datalisty")
              joinlines ("datalistx,datalisty",>"star-"//n//".reg")

              delete ("datalistx")
              delete ("datalisty")

              n=n+1
       }
       n=1

       m=2
}
;

if (xshift<0 && xshift<0-0.5*xlen){
       delete (files)
       next
}
;
if (xshift>0 && xshift>0.5*xlen){
       delete (files)
       next
}
;
if (yshift<0 && yshift<0-0.5*ylen){
       delete (files)
       next
}
;
if (yshift>0 && yshift>0.5*ylen){
       delete (files)
       next
}
;

if (phot_meth=="Flux"){
       print (3*real(s2),>>"temp_frame_fwhm_aperture_flux"//flux_percent//".info")
       print (real(s2),>>"temp_frame_fwhm_flux"//flux_percent//".info")
       print ((s10),>>"temp_frame_subset_flux"//flux_percent//".info")
       print ((s5),>>"temp_HJD.info")
}
;
if (phot_meth=="FWHM"){
       print (3*real(s2),>>"temp_frame_fwhm_aperture_FWHM"//fwhm_multiple//".info")
       print (real(s2),>>"temp_frame_fwhm_FWHM"//fwhm_multiple//".info")
       print ((s10),>>"temp_frame_subset_FWHM"//fwhm_multiple//".info")
```

```
	print ((s5),>>"temp_HJD.info")
}
;

sections ("@"//(coordinate))
for (n=1;n<=sections.nimages;n+=1){
	tabpar ((coordinate),1,n)
	xstar=real(tabpar.value)
	tabpar ((coordinate),2,n)
	ystar=real(tabpar.value)

	if (phot_meth=="Flux"){
		if (xshift<=0&&xstar>xlen+xshift-10){
			print (3*real(s2),>"star_radi-"//n)
			cont=1
			print (3*real(s2),>>"temp_radii_star-"//n//"_flux"//flux_percent//".info")
			print ((s5),>>"temp_star-"//n//"_HJD.info")
		}
		;
		if (xshift>=0&&xstar<0+xshift+10&&cont!=1){
			print (3*real(s2),>"star_radi-"//n)
			cont=1
			print (3*real(s2),>>"temp_radii_star-"//n//"_flux"//flux_percent//".info")
			print ((s5),>>"temp_star-"//n//"_HJD.info")
		}
		;
		if (xstar>xlen-10&&cont!=1){
			print (3*real(s2),>"star_radi-"//n)
			cont=1
			print (3*real(s2),>>"temp_radii_star-"//n//"_flux"//flux_percent//".info")
			print ((s5),>>"temp_star-"//n//"_HJD.info")
		}
		;
		if (xstar<10&&cont!=1){
			print (3*real(s2),>"star_radi-"//n)
			cont=1
			print (3*real(s2),>>"temp_radii_star-"//n//"_flux"//flux_percent//".info")
			print ((s5),>>"temp_star-"//n//"_HJD.info")
		}
		;
		if (yshift<=0&&ystar>ylen+yshift-10&&cont!=1){
			print (3*real(s2),>"star_radi-"//n)
			cont=1
			print (3*real(s2),>>"temp_radii_star-"//n//"_flux"//flux_percent//".info")
			print ((s5),>>"temp_star-"//n//"_HJD.info")
		}
		;
		if (yshift>=0&&ystar<0+yshift+10&&cont!=1){
			print (3*real(s2),>"star_radi-"//n)
			cont=1
			print (3*real(s2),>>"temp_radii_star-"//n//"_flux"//flux_percent//".info")
			print ((s5),>>"temp_star-"//n//"_HJD.info")
		}
		;
		if (ystar>ylen-10&&cont!=1){
			print (3*real(s2),>"star_radi-"//n)
			cont=1
			print (3*real(s2),>>"temp_radii_star-"//n//"_flux"//flux_percent//".info")
			print ((s5),>>"temp_star-"//n//"_HJD.info")
		}
		;
		if (ystar<10&&cont!=1){
			print (3*real(s2),>"star_radi-"//n)
			cont=1
			print (3*real(s2),>>"temp_radii_star-"//n//"_flux"//flux_percent//".info")
			print ((s5),>>"temp_star-"//n//"_HJD.info")
		}
		;
		if (cont==0){
			print ("q",>>"nomoreqs")

			clear
			print ("Current file",files)
			print ("Calculating PSF Flux of star: star-",n,".reg")

			psfmeasure (files,disp-,level=int (flux_percent),size="Radius",imagecur="star-
"//n//".reg",graphcur="nomoreqs",>"dataradA")
```

```
                        sed -e '/NOAO/d' -e '/Image/d' -e '/Average/d' -e '/flux/d' -e 's/" "/d/g' -e
's/obj-........../ddddddddddddddd/g' "dataradA" >> "dataradB"

                        list2="dataradB"
                        while (fscan(list2,s3)!=EOF){
                              temp=(s3)
                              i=strlen(temp)
                              if (l!=1&&l<=2){
                                    temp=substr (temp,41,i-15)
                                    print (temp,>>"dataradC")
                              }
                              ;
                              l=l+1
                        }
                        l=1

                        sed -e 's/d//g' "dataradC" >> "dataradD"
                        rename ("dataradD","star_radi-"//n)

                        print (xstar,>>"onframexstar")
                        print (ystar,>>"onframeystar")

                        tabpar ("star_radi-"//n,1,1)
                        temp=tabpar.value
                        print (temp,>>"temp_radii_star-"//n//"_flux"//flux_percent//".info")
                        print ((s5),>>"temp_star-"//n//"_HJD.info")
                  }
                  ;

                  if (n==1){
                        rename ("star_radi-1","radius.info")
                  }
                  ;
                  if (n!=1){
                        concat ("radius.info"//",","//"star_radi-"//n,>"tempradii")
                        delete ("radius.info")
                        rename ("tempradii","radius.info")
                  }
                  ;

                  delete ("datarad*")
                  cont=0
            }
            ;

            if (phot_meth=="FWHM"){
                  if (xshift<=0&&xstar>xlen+xshift-10){
                        print (3*real(s2),>"star_fwhm-"//n)
                        cont=1
                        print (3*real(s2),>>"temp_fwhm_star-"//n//"_FWHM"//fwhm_multiple//".info")
                        print ((s5),>>"temp_star-"//n//"_HJD.info")
                  }
                  ;
                  if (xshift>=0&&xstar<0+xshift+10&&cont!=1){
                        print (3*real(s2),>"star_fwhm-"//n)
                        cont=1
                        print (3*real(s2),>>"temp_fwhm_star-"//n//"_FWHM"//fwhm_multiple//".info")
                        print ((s5),>>"temp_star-"//n//"_HJD.info")
                  }
                  ;
                  if (xstar>xlen-10&&cont!=1){
                        print (3*real(s2),>"star_fwhm-"//n)
                        cont=1
                        print (3*real(s2),>>"temp_fwhm_star-"//n//"_FWHM"//fwhm_multiple//".info")
                        print ((s5),>>"temp_star-"//n//"_HJD.info")
                  }
                  ;
                  if (xstar<10&&cont!=1){
                        print (3*real(s2),>"star_fwhm-"//n)
                        cont=1
                        print (3*real(s2),>>"temp_fwhm_star-"//n//"_FWHM"//fwhm_multiple//".info")
                        print ((s5),>>"temp_star-"//n//"_HJD.info")
                  }
                  ;
                  if (yshift<=0&&ystar>ylen+yshift-10&&cont!=1){
                        print (3*real(s2),>"star_fwhm-"//n)
```

```
                                cont=1
                                print (3*real(s2),>>"temp_fwhm_star-"//n//"_FWHM"//fwhm_multiple//".info")
                                print ((s5),>>"temp_star-"//n//"_HJD.info")
                        }
                        ;
                        if (yshift>=0&&ystar<0+yshift+10&&cont!=1){
                                print (3*real(s2),>"star_fwhm-"//n)
                                cont=1
                                print (3*real(s2),>>"temp_fwhm_star-"//n//"_FWHM"//fwhm_multiple//".info")
                                print ((s5),>>"temp_star-"//n//"_HJD.info")
                        }
                        ;
                        if (ystar>ylen-10&&cont!=1){
                                print (3*real(s2),>"star_fwhm-"//n)
                                cont=1
                                print (3*real(s2),>>"temp_fwhm_star-"//n//"_FWHM"//fwhm_multiple//".info")
                                print ((s5),>>"temp_star-"//n//"_HJD.info")
                        }
                        ;
                        if (ystar<10&&cont!=1){
                                print (3*real(s2),>"star_fwhm-"//n)
                                cont=1
                                print (3*real(s2),>>"temp_fwhm_star-"//n//"_FWHM"//fwhm_multiple//".info")
                                print ((s5),>>"temp_star-"//n//"_HJD.info")
                        }
                        ;
                        if (cont==0){
                                print ("q",>>"nomoreqs")

                                clear
                                print ("Current file",files)
                                print ("Calculating PSF FWHM of star: star-",n,".reg")

                                psfmeasure (files,disp-,level=0.5,size="FWHM",imagecur="star-
"//n//".reg",graphcur="nomoreqs",>"datafwhmA")

                                sed -e '/NOAO/d' -e '/Image/d' -e '/Average/d' -e 's/" "/d/g' -e 's/obj-
.........../ddddddddddddddd/g' "datafwhmA" >> "datafwhmB"

                                list2="datafwhmB"
                                while (fscan(list2,s3)!=EOF){
                                        temp=(s3)
                                        i=strlen(temp)
                                        if (l!=1&&l<=2){
                                                temp=substr (temp,41,i-15)
                                                print (temp,>>"datafwhmC")
                                        }
                                        ;
                                        l=l+1
                                }
                                l=1

                                sed -e 's/d//g' "datafwhmC" >> "datafwhmD"
                                rename ("datafwhmD","star_fwhm-"//n)

                                print (xstar,>>"onframexstar")
                                print (ystar,>>"onframeystar")

                                tabpar ("star_fwhm-"//n,1,1)
                                temp=tabpar.value
                                print (temp,>>"temp_fwhm_star-"//n//"_FWHM"//fwhm_multiple//".info")
                                print ((s5),>>"temp_star-"//n//"_HJD.info")
                        }
                        ;

                        if (n==1){
                                rename ("star_fwhm-1","fwhm.info")
                        }
                        ;
                        if (n!=1){
                                concat ("fwhm.info"//","//"star_fwhm-"//n,>"tempfwhm")
                                delete ("fwhm.info")
                                rename ("tempfwhm","fwhm.info")
                        }
                        ;

                        delete ("datafwhm*")
```

```
            cont=0
      }
      ;
}
n=1

joinlines ("onframexstar,onframeystar",>"onframe.reg")

phot (files,coords="onframe.reg",interac-,verif-)

txdump (files//".mag.1","nsky",yes,>>"nsky.info")
type "nsky.info" | average > "sky.info"

tabpar ("sky.info",1,1)
skypix=real(tabpar.value)

delete ("*sky*")

while (skypix<500){
      dann=dann+1
      fitskypars.dannulu=dann

      clear
      print ("Determining proper annulus and dannulus"
      print ("")

      delete (files//".mag.1")
      phot (files,coords="onframe.reg",interac-,verif-)

      txdump (files//".mag.1","nsky",yes,>>"nsky.info")
      type "nsky.info" | average > "sky.info"

      tabpar ("sky.info",1,1)
      skypix=real(tabpar.value)

      delete ("*sky*")
}
while (skypix>505){
      dann=dann-0.1
      fitskypars.dannulu=dann

      clear
      print ("Determining proper annulus and dannulus"
      print ("")

      delete (files//".mag.1")
      phot (files,coords="onframe.reg",interac-,verif-)

      txdump (files//".mag.1","nsky",yes,>"nsky.info")
      type "nsky.info" | average > "sky.info"

      tabpar ("sky.info",1,1)
      skypix=real(tabpar.value)

      delete ("*sky*")
}
delete (files//".mag.1")

if (phot_meth=="Flux"){
      list2="radius.info"
}
;
if (phot_meth=="FWHM"){
      list2="fwhm.info"
}
;
while (fscan(list2,s1)!=EOF){
      photpars.apertur=real(s1)

      clear
      print ("Photting: star-"//n//".reg")
      phot (files,output=files//".mag."//n,coords="star-"//n//".reg",interac-,verif-)

      if (n==1){
            rename (files//".mag.1","magfile")
      }
      ;
```

```
                if (n!=1){
                        pconcat ("magfile"//","//files//".mag."//n,"tempmag")
                        delete ("magfile")
                        rename ("tempmag","magfile")
                }
                ;
                n=n+1
        }
        n=1

        delete (files//".mag*")

        prenumber ("magfile")

        if (phot_meth=="Flux"&&pier_side=="Yes"){
                rename ("magfile",files//"_"//pier//"_flux"//flux_percent//".mag.1")
        }
        ;
        if (phot_meth=="Flux"&&pier_side=="No"){
                rename ("magfile",files//"_NA_flux"//flux_percent//".mag.1")
        }
        ;
        if (phot_meth=="FWHM"&&pier_side=="Yes"){
                rename ("magfile",files//"_"//pier//"_FWHM"//fwhm_multiple//".mag.1")
        }
        ;
        if (phot_meth=="FWHM"&&pier_side=="No"){
                rename ("magfile",files//"_NA_FWHM"//fwhm_multiple//".mag.1")
        }
        ;

        delete ("onframe*star")
        delete ("star_*")

        if (phot_meth=="Flux"){
                delete ("radius.info")
        }
        ;
        if (phot_meth=="FWHM"){
                delete ("fwhm.info")
        }
        ;

        delete ("onframe.reg")

        dann=1
    }
}
;

if (pier_side=="Yes"){
     list1="datalistpier"
     while (fscan(list1,s1)!=EOF){
            pier=(s1)
            if (phot_meth=="Flux"){
                sections
("*"//pier//"_flux"//flux_percent//".mag*",opt="fullname",>>"datalist_"//pier//"_magfiles")
            }
            ;
            if (phot_meth=="FWHM"){
                sections
("*"//pier//"_FWHM"//fwhm_multiple//".mag*",opt="fullname",>>"datalist_"//pier//"_magfiles")
            }
            ;
            list2="datalist_"//pier//"_magfiles"
            while (fscan(list2,s2)!=EOF){
                files=(s2)

                txdump (files,"id,mag,otime,xairmass,ifilter,itime",yes,>>"star.lst")
            }

            #changes INDEFs to 30.0 since photpars.zmag = 30
            clear
            print ("Changing INDEFs to zmag = 30 in .lst files")
            sleep (1)

            sed -e 's/INDEF/30.0/g' "star.lst" > "startemp.lst"
```

```
	delete ("star.lst")
	rename ("startemp.lst","star.lst")

	fields ("star.lst",5,>"datalist_"//pier//"_filter")
	tsort ("datalist_"//pier//"_filter",1)
	list2="datalist_"//pier//"_filter"
	while (fscan(list2,s2)!=EOF){
		filter=(s2)

		if (checkfilter!=filter){
			print (filter,>>"datalist_"//pier//"_filtertable")
		}
		;
		checkfilter=filter
	}
	checkfilter=""

	fields ("star.lst",6,>"datalist_"//pier//"_inttime")
	tsort ("datalist_"//pier//"_inttime",1)
	list2="datalist_"//pier//"_inttime"
	while (fscan(list2,s2)!=EOF){
		inttime=(s2)

		if (checkinttime!=inttime){
			print (inttime,>>"datalist_"//pier//"_inttimetable")
		}
		;
		checkinttime=inttime
	}
	checkinttime=""

	list2="datalist_"//pier//"_filtertable"
	while (fscan(list2,s2)!=EOF){
		filter=(s2)
		list3="datalist_"//pier//"_inttimetable"
		while (fscan(list3,s3)!=EOF){
			inttime=(s3)

			if (phot_meth=="Flux"){
				match (filter,"star.lst",>"startemp.lst")
				match (inttime,"startemp.lst",>"startemp2.lst")
				fields ("startemp2.lst","1-
5",>"star"//filter//"_"//inttime//"_"//pier//"_flux"//flux_percent//".lst")
				sections
("@star"//filter//"_"//inttime//"_"//pier//"_flux"//flux_percent//".lst")
				if (sections.nimages==0){
					delete
("star"//filter//"_"//inttime//"_"//pier//"_flux"//flux_percent//".lst")
				}
				;
			}
			;
			if (phot_meth=="FWHM"){
				match (filter,"star.lst",>"startemp.lst")
				match (inttime,"startemp.lst",>"startemp2.lst")
				fields ("startemp2.lst","1-
5",>"star"//filter//"_"//inttime//"_"//pier//"_FWHM"//fwhm_multiple//".lst")
				sections
("@star"//filter//"_"//inttime//"_"//pier//"_FWHM"//fwhm_multiple//".lst")
				if (sections.nimages==0){
					delete
("star"//filter//"_"//inttime//"_"//pier//"_FWHM"//fwhm_multiple//".lst")
				}
				;
			}
			;
			delete ("startemp.lst")
			delete ("startemp2.lst")
		}
	}

	delete ("star.lst")

	if (phot_meth=="Flux"){
		tsort ("*_"//pier//"_flux"//flux_percent//".lst","3,1")
		sections ("*_"//pier//"_flux"//flux_percent//".lst",>>"datalistlstfiles")
	}
```

```
            ;
            if (phot_meth=="FWHM"){
                  tsort ("*_"//pier//"_FWHM"//fwhm_multiple//".lst","3,1")
                  sections ("*_"//pier//"_FWHM"//fwhm_multiple//".lst",>>"datalistlstfiles")
            }
            ;
      }
}
;
if (pier_side=="No"){
      if (phot_meth=="Flux"){
            sections ("*NA_flux"//flux_percent//".mag*",opt="fullname",>>"datalistmagfiles")
      }
      ;
      if (phot_meth=="FWHM"){
            sections ("*NA_FWHM"//fwhm_multiple//".mag*",opt="fullname",>>"datalistmagfiles")
      }
      ;

      list1="datalistmagfiles"
      while (fscan(list1,s1)!=EOF){
            files=(s1)

            txdump (files,"id,mag,otime,xairmass,ifilter,itime",yes,>>"star.lst")
      }

      #changes INDEFs to 30.0 since photpars.zmag = 30
      clear
      print ("Changing INDEFs to zmag = 30 in .lst files")
      sleep (1)

      sed -e 's/INDEF/30.0/g' "star.lst" > "startemp.lst"
      delete ("star.lst")
      rename ("startemp.lst","star.lst")

      fields ("star.lst",5,>"datalistfilter")
      tsort ("datalistfilter",1)
      list1="datalistfilter"
      while (fscan(list1,s1)!=EOF){
            filter=(s1)

            if (checkfilter!=filter){
                  print (filter,>>"datalistfiltertable")
            }
            ;
            checkfilter=filter
      }

      fields ("star.lst",6,>"datalist_inttime")
      tsort ("datalist_inttime",1)
      list1="datalist_inttime"
      while (fscan(list1,s1)!=EOF){
            inttime=(s1)

            if (checkinttime!=inttime){
                  print (inttime,>>"datalist_inttimetable")
            }
            ;
            checkinttime=inttime
      }
      checkinttime=""

      list1="datalistfiltertable"
      while (fscan(list1,s1)!=EOF){
            filter=(s1)
            list2="datalist_inttimetable"
            while (fscan(list2,s2)!=EOF){
                  inttime=(s2)
                  if (phot_meth=="Flux"){
                        match (filter,"star.lst",>"startemp.lst")
                        match (inttime,"startemp.lst",>"startemp2.lst")
                        fields ("startemp2.lst","1-
5",>"star"//filter//"_"//inttime//"_NA_flux"//flux_percent//".lst")
                        sections ("@star"//filter//"_"//inttime//"_NA_flux"//flux_percent//".lst")
                        if (sections.nimages==0){
                              delete ("star"//filter//"_"//inttime//"_NA_flux"//flux_percent//".lst")
                        }
```

```
                        ;
                  }
                  ;
                  if (phot_meth=="FWHM"){
                        match (filter,"star.lst",>"startemp.lst")
                        match (inttime,"startemp.lst",>"startemp2.lst")
                        fields ("startemp2.lst","1-
5",>"star"//filter//"_"//inttime//"_NA_FWHM"//fwhm_multiple//".lst")
                        sections ("@star"//filter//"_"//inttime//"_NA_FWHM"//fwhm_multiple//".lst")
                        if (sections.nimages==0){
                              delete ("star"//filter//"_"//inttime//"_NA_FWHM"//fwhm_multiple//".lst")
                        }
                        ;
                  }
                  ;
                  delete ("startemp.lst")
                  delete ("startemp2.lst")
            }
      }

      delete ("star.lst")

      if (phot_meth=="Flux"){
            tsort ("*_NA_flux"//flux_percent//".lst","3,1")
            sections ("*_NA_flux"//flux_percent//".lst",>>"datalistlstfiles")
      }
      ;
      if (phot_meth=="FWHM"){
            tsort ("*_NA_FWHM"//fwhm_multiple//".lst","3,1")
            sections ("*_NA_FWHM"//fwhm_multiple//".lst",>>"datalistlstfiles")
      }
      ;
}
;

clear
print ("Removing possible duplicates in .lst files")
sleep (1)

list1="datalistlstfiles"
while (fscan(list1,s1)!=EOF){
      lstfiles=(s1)
      list2=lstfiles
      while (fscan(list2,s2,s3,s4,s5,s6)!=EOF){
            star=(s2)
            magnitude=(s3)
            HJD=(s4)
            airmass=(s5)
            filter=(s6)

            if (star!=checkstar){
                  print (star,>>"datalistar")
                  print (magnitude,>>"datalistmag")
                  print (HJD,>>"datalistHJD")
                  print (airmass,>>"datalistairmass")
                  print (filter,>>"datalistfilter")
            }
            ;
            checkstar=star
      }
      joinlines ("datalistar,datalistmag,datalistHJD,datalistairmass,datalistfilter",>lstfiles//".corr")

      delete ("datalistar")
      delete ("datalistmag")
      delete ("datalistHJD")
      delete ("datalistairmass")
      delete ("datalistfilter")

      delete (lstfiles)
      rename (lstfiles//".corr",lstfiles)
}

clear
print ("Creating .info files")
print ("")
sleep (1)
```

```
sections ("@"//(coordinate))
for (n=1;n<=sections.nimages;n+=1){
     if (phot_meth=="Flux"&&pier_side=="Yes"){
           print ("Creating radii_star-"//n//"_EW_flux"//flux_percent//".info")
           joinlines ("temp_star-"//n//"_HJD.info,temp_radii_star-
"//n//"_flux"//flux_percent//".info",>"radii_star-"//n//"_EW_flux"//flux_percent//".info")
     }
     ;
     if (phot_meth=="FWHM"&&pier_side=="Yes"){
           print ("Creating fwhm_star-"//n//"_EW_FWHM"//fwhm_multiple//".info")
           joinlines ("temp_star-"//n//"_HJD.info,temp_fwhm_star-
"//n//"_FWHM"//fwhm_multiple//".info",>"fwhm_star-"//n//"_EW_FWHM"//fwhm_multiple//".info")
     }
     ;
     if (phot_meth=="Flux"&&pier_side=="No"){
           print ("Creating radii_star-"//n//"_NA_flux"//flux_percent//".info")
           joinlines ("temp_star-"//n//"_HJD.info,temp_radii_star-
"//n//"_flux"//flux_percent//".info",>"radii_star-"//n//"_NA_flux"//flux_percent//".info")
     }
     ;
     if (phot_meth=="FWHM"&&pier_side=="No"){
           print ("Creating fwhm_star-"//n//"_NA_FWHM"//fwhm_multiple//".info")
           joinlines ("temp_star-"//n//"_HJD.info,temp_fwhm_star-
"//n//"_FWHM"//fwhm_multiple//".info",>"fwhm_star-"//n//"_NA_FWHM"//fwhm_multiple//".info")
     }
     ;
}

if (phot_meth=="Flux"&&pier_side=="Yes"){
     print ("")
     print ("Creating avg frame fwhm aperture and frame fwhm .info file")
     joinlines
("temp_HJD.info,temp_frame_fwhm_aperture_flux"//flux_percent//".info,temp_frame_subset_flux"//flux_percent//".i
nfo",>"avg_frame_fwhm_aperture_EW_flux"//flux_percent//".info")
     joinlines
("temp_HJD.info,temp_frame_fwhm_flux"//flux_percent//".info,temp_frame_subset_flux"//flux_percent//".info",>"av
g_frame_fwhm_EW_flux"//flux_percent//".info")
}
;

if (phot_meth=="FWHM"&&pier_side=="Yes"){
     print ("")
     print ("Creating avg frame fwhm aperture and frame fwhm .info file")
     joinlines
("temp_HJD.info,temp_frame_fwhm_aperture_FWHM"//fwhm_multiple//".info,temp_frame_subset_FWHM"//fwhm_multiple//"
.info",>"avg_frame_fwhm_aperture_EW_FWHM"//fwhm_multiple//".info")
     joinlines
("temp_HJD.info,temp_frame_fwhm_FWHM"//fwhm_multiple//".info,temp_frame_subset_FWHM"//fwhm_multiple//".info",>"
avg_frame_fwhm_EW_FWHM"//fwhm_multiple//".info")
}
;
if (phot_meth=="Flux"&&pier_side=="No"){
     print ("")
     print ("Creating avg frame fwhm aperture and frame fwhm .info file")
     joinlines
("temp_HJD.info,temp_frame_fwhm_aperture_flux"//flux_percent//".info,temp_frame_subset_flux"//flux_percent//".i
nfo",>"avg_frame_fwhm_aperture_NA_flux"//flux_percent//".info")
     joinlines
("temp_HJD.info,temp_frame_fwhm_flux"//flux_percent//".info,temp_frame_subset_flux"//flux_percent//".info",>"av
g_frame_fwhm_NA_flux"//flux_percent//".info")
}
;
if (phot_meth=="FWHM"&&pier_side=="No"){
     print ("")
     print ("Creating avg frame fwhm aperture and frame fwhm .info file")
     joinlines
("temp_HJD.info,temp_frame_fwhm_aperture_FWHM"//fwhm_multiple//".info,temp_frame_subset_FWHM"//fwhm_multiple//"
.info",>"avg_frame_fwhm_aperture_NA_FWHM"//fwhm_multiple//".info")
     joinlines
("temp_HJD.info,temp_frame_fwhm_FWHM"//fwhm_multiple//".info,temp_frame_subset_FWHM"//fwhm_multiple//".info",>"
avg_frame_fwhm_NA_FWHM"//fwhm_multiple//".info")
}
;

#deletes possibly incompatible datalists and other files
delete ("*temp*")
delete ("star-*")
```

```
delete ("*.comp")
delete ("nomoreqs")

#adds date and time of when script was applied to frame headers
date '+DATE:%m/%d/%y%nTIME:%H:%M:%S' > "datalist"
tabpar ("datalist",1,1)
datestring=tabpar.value
tabpar ("datalist",1,2)
timestring=tabpar.value

hedit ("*"//searchalignphotometry//"*.fits","AP_APPHOT",datestring//""//timestring,add+,ver-)

#deletes possibly incompatible datalists
delete ("data*")

beep
```

A.2 Additional Scripts

A.2.1 *ap_reg_apphot*

```
#DOCUMENTATION: ap_reg_apphot_v_1.cl
#    ap_reg_apphot_v_1.cl by Paul Iverson

procedure ap_reg_apphot_v_1
(search_align_phot,coordinate_file,para_method,phot_method,pier_side_method,phot_aperture,fwhm_mult)

string search_align_phot {prompt="Enter search parameters for the aligned frames to phot (i.e. a-*star*B)"}
string coordinate_file   {prompt="Enter the name of the coordinate file (i.e. ds9.reg)"}
string para_method       {prompt="Enter the parameter method to be used",enum="Trad|New"}
string phot_method       {prompt="Enter the phot method to be used",enum="SetAP|FWHM"}
string pier_side_method  {prompt="Phot frames by pier side",enum="Yes|No"}
string phot_aperture     {prompt="Enter the phot aperture to be used"}
string fwhm_mult         {prompt="Enter the multiple of fwhm to be used as the phot
                         aperture",enum="1.0|1.5|2.0|2.5|3.0"}

begin
     string searchalignphotometry
     string coordinate
     string para_meth
     string phot_meth
     string phot_apert
     string pier_side
     string fwhm_multiple
end

#variable declarations
int         n=1
real        skypix
real        dann=1
string      s1=""
string      s2=""
string      s3=""
string      s4=""
string      s5=""
string      s6=""
string      s7=""
string      s8=""
string      s9=""
string      s10=""
string      files
string      filter=""
string      checkfilter=""
string      inttime=""
string      checkinttime=""
string      lstfiles=""
string      star=""
string      checkstar=""
string      magnitude=""
string      HJD=""
string      airmass=""
string      pier=""
string      checkpier=""
string      X=""
string      Y=""
struct      *list1
struct      *list2
struct      *list3

#deletes possibly incompatible datalists
delete ("data*")

searchalignphotometry=search_align_phot
coordinate=coordinate_file
para_meth=para_method
phot_meth=phot_method
if (phot_meth=="SetAP"){
     phot_apert=phot_aperture
}
;
if (phot_meth=="FWHM"){
     fwhm_multiple=fwhm_mult
}
;
pier_side=pier_side_method

if (pier_side=="Yes"){
     sections ((searchalignphotometry)//"*.fits",opt="fullname",>>"datalistfile")
```

```
     if (sections.nimages>0){
           hselect
((searchalignphotometry)//"*.fits","$I,FWHM,GAIN,SKYDEV,HJD,RDNOISE,X_PIXEL_,Y_PIXEL_,PIER,SUBSET",yes,>>"datat
otal")
           tsort ("datatotal",5)
     }
     ;

     hselect ((searchalignphotometry)//"*.fits","PIER",yes,>>"datapier")
     tsort ("datapier",1)
     list1="datapier"
     while (fscan(list1,s1)!=EOF){
           pier=(s1)
           if (pier!=checkpier){
                 print (pier,>>"datalistpier")
           }
           ;
           checkpier=pier
     }

     #datalist contains files, fwhmpsf, epadu, sigma, HJD, rdnoise, pixel shifts, pier, subset
     sections @datatotal
     if (sections.nimages>0){
           list1="datatotal"
           while (fscan(list1,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10) != EOF){
                 clear
                 print ("File: ",s1)
                 print ("HJD: ",s5)
                 print ("")
                 print ("Subset: ",s10)
                 print ("")
                 print ("Pier: ",s9)
                 print ("")
                 print ("FWHM: ",s2)
                 print ("")
                 print ("Gain: ",s3)
                 print ("RDNoise: ",s6)
                 print ("Sky Deviation: ",s4)
                 print ("")
                 print ("X Pixel Shift: ",s7)
                 print ("Y Pixel Shift: ",s8)

                 sleep (1)

                 if (para_meth=="Trad"){
                       centerpars.calgorithm="centroid"
                       centerpars.cbox=13.
                       centerpars.cthreshold=0.75
                       centerpars.minsnratio=1.0
                       centerpars.cmaxiter=10
                       centerpars.maxshift=8.5
                       centerpars.clean=no
                       centerpars.rclean=1.0
                       centerpars.rclip=2.0
                       centerpars.kclean=3.0
                       centerpars.mkcenter=no

                       fitskypars.salgorithm="ofilter"
                       fitskypars.annulus=13.0
                       fitskypars.dannulus=4.0
                       fitskypars.skyvalue=0.0
                       fitskypars.smaxiter=10
                       fitskypars.sloclip=0.0
                       fitskypars.shiclip=10.0
                       fitskypars.snreject=50
                       fitskypars.sloreject=3.0
                       fitskypars.shireject=3.0
                       fitskypars.khist=3.0
                       fitskypars.binsize=0.1
                       fitskypars.smooth=no
                       fitskypars.rgrow=0.0
                       fitskypars.mksky=no

                       datapars.scale = 1.0
                       datapars.fwhmpsf = 3.0
                       datapars.emission = yes
                       datapars.sigma = INDEF
```

```
        datapars.datamin = INDEF
        datapars.datamax = INDEF
        datapars.noise = "poisson"
        datapars.ccdread = "rdnoise"
        datapars.gain = "gain"
        datapars.readnoise = INDEF
        datapars.epadu = 1.0
        datapars.exposure = "exptime"
        datapars.airmass = "airmass"
        datapars.filter = "subset"
        datapars.obstime = "hjd"
        datapars.itime = INDEF
        datapars.xairmass = INDEF
        datapars.ifilter = "INDEF"
        datapars.otime = "INDEF"

        photpars.weighting = "constant"
        photpars.mkapert = no
}
;
if (para_meth=="New"){
        centerpars.calg="centroid"
        centerpars.cbox=3*real(s2)
        centerpars.cthreshold=0.75
        centerpars.minsnratio=1.0
        centerpars.cmaxiter=20
        centerpars.maxshift=8.5
        centerpars.clean=no
        centerpars.rclean=1.0
        centerpars.rclip=2.0
        centerpars.kclean=3.0
        centerpars.mkcenter=no

        fitskypars.salgorithm="mode"
        fitskypars.annulus=10*real(s2)
        fitskypars.dannulu=1
        fitskypars.skyvalue=0.0
        fitskypars.smaxiter=10
        fitskypars.sloclip=0.0
        fitskypars.shiclip=10.0
        fitskypars.snreject=50
        fitskypars.sloreject=3.0
        fitskypars.shireject=3.0
        fitskypars.khist=3.0
        fitskypars.binsize=0.1
        fitskypars.smooth=no
        fitskypars.rgrow=0.0
        fitskypars.mksky=no

        datapars.scale = 1.0
        datapars.fwhmpsf=real(s2)
        datapars.emission = yes
        datapars.sigma=real(s4)
        datapars.datamin=INDEF
        datapars.datamax=INDEF
        datapars.noise = "poisson"
        datapars.ccdread="RDNOISE"
        datapars.gain="GAIN"
        datapars.readnoise=real(s6)
        datapars.epadu=real(s3)
        datapars.exposure="EXPTIME"
        datapars.airmass="AIRMASS"
        datapars.filter="SUBSET"
        datapars.obstime="HJD"
        datapars.itime = INDEF
        datapars.xairmass = INDEF
        datapars.ifilter = "INDEF"
        datapars.otime = "INDEF"

        photpars.weighting = "constant"
        photpars.mkapert = no
}
;

photpars.weighting="constant"
if (phot_meth=="SetAP"){
        photpars.apertures=real(phot_apert)
```

```
                }
                ;
                if (phot_meth=="FWHM"){
                      photpars.apertures=real(fwhm_multiple)*real(s2)
                }
                ;
                photpars.zmag=30

                psfmeasure.radius=5
                psfmeasure.sbuffer=5
                psfmeasure.swidth=5

                files=(s1)

                clear
                print ("Photting file: ",files)

                if (phot_meth=="SetAP"){
                      phot
(files,coords=coordinate,output=files//"_"//s9//"_aperture"//phot_apert//".mag.1",interac-,verif-)
                }
                ;
                if (phot_meth=="FWHM"){
                      phot
(files,coords=coordinate,output=files//"_"//s9//"_apFWHM"//fwhm_multiple//".mag.1",interac-,verif-)
                }
                ;

                if (para_meth=="New"){
                      if (phot_meth=="SetAP"){
                            txdump
(files//"_"//s9//"_aperture"//phot_apert//".mag.1","nsky",yes,>>"nsky.info")
                      }
                      ;
                      if (phot_meth=="FWHM"){
                            txdump
(files//"_"//s9//"_apFWHM"//fwhm_multiple//".mag.1","nsky",yes,>>"nsky.info")
                      }
                      ;
                      type "nsky.info" | average > "sky.info"

                      tabpar ("sky.info",1,1)
                      skypix=real(tabpar.value)

                      delete ("*sky*")

                      while (skypix<500){
                            dann=dann+1
                            fitskypars.dannulu=dann

                            clear
                            print ("Determining proper annulus and dannulus"
                            print ("")

                            if (phot_meth=="SetAP"){
                                  delete (files//"_"//s9//"_aperture"//phot_apert//".mag.1")
                                  phot
(files,coords=coordinate,output=files//"_"//s9//"_aperture"//phot_apert//".mag.1",interac-,verif-)
                                  txdump
(files//"_"//s9//"_aperture"//phot_apert//".mag.1","nsky",yes,>>"nsky.info")
                            }
                            ;
                            if (phot_meth=="FWHM"){
                                  delete (files//"_"//s9//"_apFWHM"//fwhm_multiple//".mag.1")
                                  phot
(files,coords=coordinate,output=files//"_"//s9//"_apFWHM"//fwhm_multiple//".mag.1",interac-,verif-)
                                  txdump
(files//"_"//s9//"_apFWHM"//fwhm_multiple//".mag.1","nsky",yes,>>"nsky.info")
                            }
                            ;
                            type "nsky.info" | average > "sky.info"

                            tabpar ("sky.info",1,1)
                            skypix=real(tabpar.value)

                            delete ("*sky*")
                      }
```

```
                        while (skypix>505){
                                dann=dann-0.1
                                fitskypars.dannulu=dann

                                clear
                                print ("Determining proper annulus and dannulus"
                                print ("")

                                if (phot_meth=="SetAP"){
                                        delete (files//"_"//s9//"_aperture"//phot_apert//".mag.1")
                                        phot
(files,coords=coordinate,output=files//"_"//s9//"_aperture"//phot_apert//".mag.1",interac-,verif-)
                                        txdump
(files//"_"//s9//"_aperture"//phot_apert//".mag.1","nsky",yes,>>"nsky.info")
                                }
                                ;
                                if (phot_meth=="FWHM"){
                                        delete (files//"_"//s9//"_apFWHM"//fwhm_multiple//".mag.1")
                                        phot
(files,coords=coordinate,output=files//"_"//s9//"_apFWHM"//fwhm_multiple//".mag.1",interac-,verif-)
                                        txdump
(files//"_"//s9//"_apFWHM"//fwhm_multiple//".mag.1","nsky",yes,>>"nsky.info")
                                }
                                ;
                                type "nsky.info" | average > "sky.info"

                                tabpar ("sky.info",1,1)
                                skypix=real(tabpar.value)

                                delete ("*sky*")
                        }
                }
                ;
        }
    }
    ;

    list1="datalistpier"
    while (fscan(list1,s1)!=EOF){
        pier=(s1)

        if (phot_meth=="SetAP"){
            sections
("*"//pier//"_aperture"//phot_apert//".mag.1",opt="fullname",>>"datalist_"//pier//"_magfiles")
        }
        ;
        if (phot_meth=="FWHM"){
            sections
("*"//pier//"_apFWHM"//fwhm_multiple//".mag.1",opt="fullname",>>"datalist_"//pier//"_magfiles")
        }
        ;

        list2="datalist_"//pier//"_magfiles"
        while (fscan(list2,s2) != EOF){
            files=(s2)

            txdump (files,"id,mag,otime,xairmass,ifilter,itime",yes,>>"star.lst")
        }

        #changes INDEFs to 30.0 since photpars.zmag = 30
        clear
        print ("Changing INDEFs to zmag = 30 in .lst files")
        sleep (1)

        sed -e 's/INDEF/30.0/g' "star.lst" > "startemp.lst"
        delete ("star.lst")
        rename ("startemp.lst","star.lst")

        fields ("star.lst",5,>"datalist_"//pier//"_filter")
        tsort ("datalist_"//pier//"_filter",1)
        list2="datalist_"//pier//"_filter"
        while (fscan(list2,s2)!=EOF){
            filter=(s2)

            if (checkfilter!=filter){
                print (filter,>>"datalist_"//pier//"_filtertable")
            }
```

```
                ;
                checkfilter=filter
            }
            checkfilter=""
            fields ("star.lst",6,>"datalist_"//pier//"_inttime")
            tsort ("datalist_"//pier//"_inttime",1)
            list2="datalist_"//pier//"_inttime"
            while (fscan(list2,s2)!=EOF){
                inttime=(s2)

                if (checkinttime!=inttime){
                    print (inttime,>>"datalist_"//pier//"_inttimetable")
                }
                ;
                checkinttime=inttime
            }
            checkinttime=""

            list2="datalist_"//pier//"_filtertable"
            while (fscan(list2,s2)!=EOF){
                filter=(s2)
                list3="datalist_"//pier//"_inttimetable"
                while (fscan(list3,s3)!=EOF){
                    inttime=(s3)

                    match (filter,"star.lst",>"startemp.lst")
                    match (inttime,"startemp.lst",>"startemp2.lst")
                    if (phot_meth=="SetAP"){
                        fields ("startemp2.lst","1-
5",>"star"//filter//"_"//inttime//"_"//pier//"_ap"//phot_apert//".lst")
                        sections ("@star"//filter//"_"//inttime//"_"//pier//"_ap"//phot_apert//".lst")
                        if (sections.nimages==0){
                            delete
("star"//filter//"_"//inttime//"_"//pier//"_ap"//phot_apert//".lst")
                        }
                        ;
                    }
                    ;
                    if (phot_meth=="FWHM"){
                        fields ("startemp2.lst","1-
5",>"star"//filter//"_"//inttime//"_"//pier//"_apFWHM"//fwhm_multiple//".lst")
                        sections
("@star"//filter//"_"//inttime//"_"//pier//"_apFWHM"//fwhm_multiple//".lst")
                        if (sections.nimages==0){
                            delete
("star"//filter//"_"//inttime//"_"//pier//"_apFWHM"//fwhm_multiple//".lst")
                        }
                        ;
                    }
                    ;

                    delete ("startemp.lst")
                    delete ("startemp2.lst")
                }
            }

            delete ("star.lst")

            if (phot_meth=="SetAP"){
                tsort ("*_"//pier//"_ap"//phot_apert//".lst","3,1")
                sections ("*_"//pier//"_ap"//phot_apert//".lst",>>"datalistlstfiles")
            }
            ;
            if (phot_meth=="FWHM"){
                tsort ("*_"//pier//"_apFWHM"//fwhm_multiple//".lst","3,1")
                sections ("*_"//pier//"_apFWHM"//fwhm_multiple//".lst",>>"datalistlstfiles")
            }
            ;
        }
}
;

if (pier_side=="No"){
     sections ((searchalignphotometry)//"*.fits",opt="fullname",>>"datalistfile")
     if (sections.nimages>0){
```

```
          hselect
((searchalignphotometry)//"*.fits","$I,FWHM,GAIN,SKYDEV,HJD,RDNOISE,X_PIXEL_,Y_PIXEL_,SUBSET",yes,>>"datatotal"
)
          tsort ("datatotal",5)
     }
     ;

     #datalist contains files, fwhmpsf, epadu, sigma, HJD, rdnoise, pixel shifts, subset
     sections ("@datatotal")
     if (sections.nimages>0){
          list1="datatotal"
          while (fscan(list1,s1,s2,s3,s4,s5,s6,s7,s8,s9) != EOF){
               clear
               print ("File: ",s1)
               print ("HJD: ",s5)
               print ("")
               print ("Subset: ",s9)
               print ("")
               print ("FWHM: ",s2)
               print ("")
               print ("Gain: ",s3)
               print ("RDNoise: ",s6)
               print ("Sky Deviation: ",s4)
               print ("")
               print ("X Pixel Shift: ",s7)
               print ("Y Pixel Shift: ",s8)

               sleep (1)

               if (para_meth=="Trad"){
                    centerpars.calgorithm="centroid"
                    centerpars.cbox=13.
                    centerpars.cthreshold=0.75
                    centerpars.minsnratio=1.0
                    centerpars.cmaxiter=10
                    centerpars.maxshift=8.5
                    centerpars.clean=no
                    centerpars.rclean=1.0
                    centerpars.rclip=2.0
                    centerpars.kclean=3.0
                    centerpars.mkcenter=no

                    fitskypars.salgorithm="ofilter"
                    fitskypars.annulus=13.0
                    fitskypars.dannulus=4.0
                    fitskypars.skyvalue=0.0
                    fitskypars.smaxiter=10
                    fitskypars.sloclip=0.0
                    fitskypars.shiclip=10.0
                    fitskypars.snreject=50
                    fitskypars.sloreject=3.0
                    fitskypars.shireject=3.0
                    fitskypars.khist=3.0
                    fitskypars.binsize=0.1
                    fitskypars.smooth=no
                    fitskypars.rgrow=0.0
                    fitskypars.mksky=no

                    datapars.scale = 1.0
                    datapars.fwhmpsf = 3.0
                    datapars.emission = yes
                    datapars.sigma = INDEF
                    datapars.datamin = INDEF
                    datapars.datamax = INDEF
                    datapars.noise = "poisson"
                    datapars.ccdread = "rdnoise"
                    datapars.gain = "gain"
                    datapars.readnoise = INDEF
                    datapars.epadu = 1.0
                    datapars.exposure = "exptime"
                    datapars.airmass = "airmass"
                    datapars.filter = "subset"
                    datapars.obstime = "hjd"
                    datapars.itime = INDEF
                    datapars.xairmass = INDEF
                    datapars.ifilter = "INDEF"
                    datapars.otime = "INDEF"
```

```
        photpars.weighting = "constant"
        photpars.mkapert = no
}
;
if (para_meth=="New"){
        centerpars.calg="centroid"
        centerpars.cbox=3*real(s2)
        centerpars.cthreshold=0.75
        centerpars.minsnratio=1.0
        centerpars.cmaxiter=20
        centerpars.maxshift=8.5
        centerpars.clean=no
        centerpars.rclean=1.0
        centerpars.rclip=2.0
        centerpars.kclean=3.0
        centerpars.mkcenter=no

        fitskypars.salgorithm="mode"
        fitskypars.annulus=10*real(s2)
        fitskypars.dannulu=1
        fitskypars.skyvalue=0.0
        fitskypars.smaxiter=10
        fitskypars.sloclip=0.0
        fitskypars.shiclip=10.0
        fitskypars.snreject=50
        fitskypars.sloreject=3.0
        fitskypars.shireject=3.0
        fitskypars.khist=3.0
        fitskypars.binsize=0.1
        fitskypars.smooth=no
        fitskypars.rgrow=0.0
        fitskypars.mksky=no

        datapars.scale = 1.0
        datapars.fwhmpsf=real(s2)
        datapars.emission = yes
        datapars.sigma=real(s4)
        datapars.datamin=INDEF
        datapars.datamax=INDEF
        datapars.noise = "poisson"
        datapars.ccdread="RDNOISE"
        datapars.gain="GAIN"
        datapars.readnoise=real(s6)
        datapars.epadu=real(s3)
        datapars.exposure="EXPTIME"
        datapars.airmass="AIRMASS"
        datapars.filter="SUBSET"
        datapars.obstime="HJD"
        datapars.itime = INDEF
        datapars.xairmass = INDEF
        datapars.ifilter = "INDEF"
        datapars.otime = "INDEF"

        photpars.weighting = "constant"
        photpars.mkapert = no
}
;

photpars.weighting="constant"
if (phot_meth=="SetAP"){
        photpars.apertures=real(phot_apert)
}
;
if (phot_meth=="FWHM"){
        photpars.apertures=real(fwhm_multiple)*real(s2)
}
;
photpars.zmag=30

psfmeasure.radius=5
psfmeasure.sbuffer=5
psfmeasure.swidth=5

files=(s1)

clear
```

```
                print ("Photting file: ",files)

                if (phot_meth=="SetAP"){
                        phot
(files,coords=coordinate,output=files//"_NA_aperture"//phot_apert//".mag.1",interac-,verif-)
                }
                ;
                if (phot_meth=="FWHM"){
                        phot
(files,coords=coordinate,output=files//"_NA_apFWHM"//fwhm_multiple//".mag.1",interac-,verif-)
                }
                ;

                if (para_meth=="New"){
                        if (phot_meth=="SetAP"){
                                txdump (files//"_NA_aperture"//phot_apert//".mag.1","nsky",yes,>>"nsky.info")
                        }
                        ;
                        if (phot_meth=="FWHM"){
                                txdump (files//"_NA_apFWHM"//fwhm_multiple//".mag.1","nsky",yes,>>"nsky.info")
                        }
                        ;
                        type "nsky.info" | average > "sky.info"

                        tabpar ("sky.info",1,1)
                        skypix=real(tabpar.value)

                        delete ("*sky*")

                        while (skypix<500){
                                dann=dann+1
                                fitskypars.dannulu=dann

                                clear
                                print ("Determining proper annulus and dannulus"
                                print ("")

                                if (phot_meth=="SetAP"){
                                        delete (files//"_NA_aperture"//phot_apert//".mag.1")
                                        phot
(files,coords=coordinate,output=files//"_NA_aperture"//phot_apert//".mag.1",interac-,verif-)
                                        txdump
(files//"_NA_aperture"//phot_apert//".mag.1","nsky",yes,>>"nsky.info")
                                }
                                ;
                                if (phot_meth=="FWHM"){
                                        delete (files//"_NA_apFWHM"//fwhm_multiple//".mag.1")
                                        phot
(files,coords=coordinate,output=files//"_NA_apFWHM"//fwhm_multiple//".mag.1",interac-,verif-)
                                        txdump
(files//"_NA_apFWHM"//fwhm_multiple//".mag.1","nsky",yes,>>"nsky.info")
                                }
                                ;
                                type "nsky.info" | average > "sky.info"

                                tabpar ("sky.info",1,1)
                                skypix=real(tabpar.value)

                                delete ("*sky*")
                        }
                        while (skypix>505){
                                dann=dann-0.1
                                fitskypars.dannulu=dann

                                clear
                                print ("Determining proper annulus and dannulus"
                                print ("")

                                if (phot_meth=="SetAP"){
                                        delete (files//"_NA_aperture"//phot_apert//".mag.1")
                                        phot
(files,coords=coordinate,output=files//"_NA_aperture"//phot_apert//".mag.1",interac-,verif-)
                                        txdump
(files//"_NA_aperture"//phot_apert//".mag.1","nsky",yes,>>"nsky.info")
                                }
                                ;
                                if (phot_meth=="FWHM"){
```

```
                                    delete (files//"_NA_apFWHM"//fwhm_multiple//".mag.1")
                                    phot
(files,coords=coordinate,output=files//"_NA_apFWHM"//fwhm_multiple//".mag.1",interac-,verif-)
                                    txdump
(files//"_NA_apFWHM"//fwhm_multiple//".mag.1","nsky",yes,>>"nsky.info")
                               }
                               ;
                               type "nsky.info" | average > "sky.info"

                               tabpar ("sky.info",1,1)
                               skypix=real(tabpar.value)

                               delete ("*sky*")
                         }
                   }
                   ;
             }
      }
      ;

      if (phot_meth=="SetAP"){
            sections ("*_NA_aperture"//phot_apert//".mag.1",opt="fullname",>>"datalist_magfiles")
      }
      ;
      if (phot_meth=="FWHM"){
            sections ("*_NA_apFWHM"//fwhm_multiple//".mag.1",opt="fullname",>>"datalist_magfiles")
      }
      ;

      list2="datalist_magfiles"
      while (fscan(list2,s2) != EOF){
            files=(s2)

            txdump (files,"id,mag,otime,xairmass,ifilter,itime",yes,>>"star.lst")
      }

      #changes INDEFs to 30.0 since photpars.zmag = 30
      clear
      print ("Changing INDEFs to zmag = 30 in .lst files")
      sleep (1)

      sed -e 's/INDEF/30.0/g' "star.lst" > "startemp.lst"
      delete ("star.lst")
      rename ("startemp.lst","star.lst")

      fields ("star.lst",5,>"datalist_filter")
      tsort ("datalist_filter",1)
      list2="datalist_filter"
      while (fscan(list2,s2)!=EOF){
            filter=(s2)

            if (checkfilter!=filter){
                  print (filter,>>"datalist_filtertable")
            }
            ;
            checkfilter=filter
      }
      checkfilter=""
      fields ("star.lst",6,>"datalist_inttime")
      tsort ("datalist_inttime",1)
      list2="datalist_inttime"
      while (fscan(list2,s2)!=EOF){
            inttime=(s2)

            if (checkinttime!=inttime){
                  print (inttime,>>"datalist_inttimetable")
            }
            ;
            checkinttime=inttime
      }
      checkinttime=""

      list2="datalist_filtertable"
      while (fscan(list2,s2)!=EOF){
            filter=(s2)
            list3="datalist_inttimetable"
            while (fscan(list3,s3)!=EOF){
```

```
                inttime=(s3)

                match (filter,"star.lst",>"startemp.lst")
                match (inttime,"startemp.lst",>"startemp2.lst")
                if (phot_meth=="SetAP"){
                        fields ("startemp2.lst","1-
5",>"star"//filter//"_"//inttime//"_NA_ap"//phot_apert//".lst")
                        sections ("@star"//filter//"_"//inttime//"_NA_ap"//phot_apert//".lst")
                        if (sections.nimages==0){
                                delete ("star"//filter//"_"//inttime//"_NA_ap"//phot_apert//".lst")
                        }
                        ;
                }
                ;
                if (phot_meth=="FWHM"){
                        fields ("startemp2.lst","1-
5",>"star"//filter//"_"//inttime//"_NA_apFWHM"//fwhm_multiple//".lst")
                        sections ("@star"//filter//"_"//inttime//"_NA_apFWHM"//fwhm_multiple//".lst")
                        if (sections.nimages==0){
                                delete ("star"//filter//"_"//inttime//"_NA_apFWHM"//fwhm_multiple//".lst")
                        }
                        ;
                }
                ;

                delete ("startemp.lst")
                delete ("startemp2.lst")
        }
    }

    delete ("star.lst")

    if (phot_meth=="SetAP"){
        tsort ("*_NA_ap"//phot_apert//".lst","3,1")
        sections ("*_NA_ap"//phot_apert//".lst",>>"datalistlstfiles")
    }
    ;
    if (phot_meth=="FWHM"){
        tsort ("*_NA_apFWHM"//fwhm_multiple//".lst","3,1")
        sections ("*_NA_apFWHM"//fwhm_multiple//".lst",>>"datalistlstfiles")
    }
    ;
}
;


clear
print ("Removing possible duplicates in .lst files")
sleep (1)

list1="datalistlstfiles"
while (fscan(list1,s1)!=EOF){
    lstfiles=(s1)
    list2=lstfiles
    while (fscan(list2,s2,s3,s4,s5,s6)!=EOF){
        star=(s2)
        magnitude=(s3)
        HJD=(s4)
        airmass=(s5)
        filter=(s6)

        if (star!=checkstar){
            print (star,>>"datalistst ar")
            print (magnitude,>>"datalistmag")
            print (HJD,>>"datalistHJD")
            print (airmass,>>"datalistairmass")
            print (filter,>>"datalistfilter")
        }
        ;
        checkstar=star
    }
    joinlines ("datalistst ar,datalistmag,datalistHJD,datalistairmass,datalistfilter",>lstfiles//".corr")

    delete ("datalistst ar")
    delete ("datalistmag")
    delete ("datalistHJD")
    delete ("datalistairmass")
```

```
     delete ("datalistfilter")

     delete (lstfiles)
     rename (lstfiles//".corr",lstfiles)
}

#deletes possibly incompatible datalists
delete ("data*")
delete ("*fits.mag*")

beep
```

A.2.2 *ap_imagecheck*

```
#apimagecheck.cl by Paul Iverson
#    walks through image frames and asks if image is wanted for photometry
#    works with both aligned and unaligned images; suggested to use both
#    before and after alignment

procedure ap_imagecheck (search_method,search_parameters,delete_file,delete_file_num)

string search_method           {prompt="Enter search method to use",enum="Single|Multiple"}
string search_parameters {prompt="Enter search parameters for the files to review (i.e. star*B)"}
string delete_file             {prompt="Delete file",enum="Yes|No|Finish"}
string delete_file_num         {prompt="Enter the number of the file to
delete",enum="1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|No|Finish"}

begin
     string search_meth
     string search_par
     string del_file=""
     string del_num
     search_meth=search_method
     search_par=search_parameters
end

int         m=1
int         n=1
int         l=1
int         totalnumimages
int         ds9loc=1
string      s1=""
string      s2=""
string      s3=""
string      s4=""
string      image=""
string      imagenum=""
struct      *list1
struct      *list2

#deletes possibly incompatible datalists
del data*

if (search_meth=="Single"){
     clear
     sections ((search_par)//"*.fits",>"datalistsearch")
     if (sections.nimages>0){
            totalnumimages=sections.nimages
            for (n=1;n<=totalnumimages;n+=1){
                  tabpar ("datalistsearch",1,n)
                  image=tabpar.value
                  disp (image,1)

                  del_file=delete_file
                  if (del_file=="Yes"){
                        del (image)
                  }
                  ;
                  if (del_file=="Finish"){
                        break
                  }
                  ;
            }
     }
     ;
}
;
if (search_meth=="Multiple"){
     clear
     print ("Set ds9 to tile images (Frame -> Tile)")
     sleep (2)

     clear
     sections ((search_par)//"*.fits",>"datalistsearch")
     if (sections.nimages>0){
            totalnumimages=sections.nimages
            for (n=1;n<=totalnumimages;n+=1){
                  tabpar ("datalistsearch",1,n)
                  image=tabpar.value

                  print (l,>>"datalistnum")
```

```
                        l=l+1
                        print (image,>>"datalistimg")

                        disp (image,ds9loc)

                        ds9loc=ds9loc+1
                        if (ds9loc==17||n==totalnumimages){
                               clear
                               del_file=delete_file

                               joinlines ("datalistnum,datalistimg",>"datalistcheck")
                               del "datalistnum"
                               del "datalistimg"

                               while (del_file=="Yes"){
                                      clear
                                      list1="datalistcheck"
                                      while (fscan(list1,s1,s2)!=EOF){
                                             print ("["//s1//"]    "//s2)
                                      }
                                      print ("")
                                      del_num=delete_file_num
                                      list1="datalistcheck"
                                      while (fscan(list1,s1,s2)!=EOF){
                                             imagenum=(s1)
                                             image=(s2)
                                             if (del_num!="Finish"&&del_num!="No"){
                                                    if (imagenum==del_num){
                                                           del (image)
                                                           ap_imagecheck.delete_file_num=""
                                                    }
                                                    ;
                                             }
                                             ;
                                             if (s1!=del_num){
                                                    print (s1,>>"datalistnum")
                                                    print (s2,>>"datalistimg")
                                             }
                                             ;
                                      }

                                      del "datalistcheck"
                                      joinlines ("datalistnum,datalistimg",>"datalistcheck")

                                      if (del_num=="No"){
                                             del_file="No"
                                      }
                                      ;
                                      if (del_num=="Finish"){
                                             del_file="Finish"
                                      }
                                      ;
                               }
                               l=1
                               ds9loc=1

                               del "datalistnum"
                               del "datalistimg"
                               del "datalistcheck"
                        }
                        ;
                        if (del_file=="Finish"){
                               break
                        }
                        ;
                 }
          }
          ;
}
;

del data*
```

A.2.3 *ap_subtrim*

```
#DOCUMENTATION: ap_subtrim_v_2.cl
#    ap_subtrim_v_2.cl by Paul Iverson
#
#           ap_subtrim_v_2.cl is a script designed to trim object frames rfits'ed with
#           the ap_rfits_v_2.cl script.
#
#           This script trims frames to user defined dimensions. Used primarily to remove large unusable
#           saturated stars.
#
#           IRAF packages required
#                  imutil
#                  astutil
#
#           Linux commands required
#                  date
#
#           Possible sources of error:
#                  Inaccurate filenames or filter entries in header -> no data in datalists

procedure ap_subtrim_v_2 (xsmall,xlarge,ysmall,ylarge)

string xsmall     {prompt="Minimum x value"}
string xlarge     {prompt="Maximum x value"}
string ysmall     {prompt="Minimum y value"}
string ylarge     {prompt="Maximum y value"}

begin
     string x1,x2,y1,y2
end

#variable declarations
int         n=1
int         xlen,ylen
string      s1=""
string      dimfile
struct      *list1

#deletes possibly incompatible datalists
del data*

sections ("*obj*fits",opt="fullname",>"datalist1")
match ("sub","datalist1",stop+,>"datalistfile")
del datalist1
tabpar ("datalistfile",1,1)
dimfile=tabpar.value

imgets (dimfile,param="i_naxis1")
xlen=int(imgets.value)
imgets (dimfile,param="i_naxis2")
ylen=int(imgets.value)

print ("Maximum x value of frame ",xlen)
x1=xsmall
x2=xlarge

print ("")
print ("Maximum y value of frame ",ylen)
y1=ysmall
y2=ylarge

#copies smaller field of frame to remove non-desired areas
imcopy ("obj-*["//x1//":"//x2//","//y1//":"//y2//"]","obj-*//.sub")

#deletes possibly incompatible datalists
del data*

#adds date and time of when script was applied to frames to headers
date '+DATE:%m/%d/%y%nTIME:%H:%M:%S' > "datalist"
tabpar ("datalist",1,1)
datestring=tabpar.value
tabpar ("datalist",1,2)
timestring=tabpar.value
hedit ("*.fits","AP_SUBTRIM",datestring//" "//timestring,add+,ver-)

#deletes possibly incompatible datalists
del data*
beep
```