IRAF MANUAL

by Thijs Coenen and Yan Grange

(2006 09 25 not the final version)

B: *Your logic does not resemble our Earth logic.*

X: *Mine is much more advanced.*

Contents

Chapter 1

Introduction

IRAF is archaic yet popular piece of software used in astrophysics. This manual describes the basics of using IRAF and some of the more specific steps you go through for spectography and photometry. The way this manual is set up we use `this typeface` for terminal output and textfiles and **`this typeface`** for the things that you have to type. In our experiences using IRAF can be quite frustrating at times, we hope this manual will alleviate that problem a bit. This manual assumes you are using the FITS file format for your image data (see chapter 3) and not `.imh` and `.pix` files.

1.1 Acknowledgements

We would like to thank Klaas, Diego, Roald, Huib, Arjan and Rens for helping us along.

Chapter 2

The basics

In this chapter we will walk you through the basics of using IRAF. It is assumed that IRAF is already set up to run on your user account. If this is not the case please have a look at the appendix. IRAF consists of set of different packages that you interact with by typing commands. A graphical version of IRAF is in the making but at the time of this writing it is not yet available. This manual has been written and tested on version IRAF V2.12EXPORT, sometimes commands change between releases. Another assumption is that you use the FITS image format, section 2.1 shows you how to set the default file format to the FITS image format.

2.1 Starting and stopping IRAF

To start IRAF you will first have to move to your IRAF directory (use the `cd` command in a terminal). Parts of IRAF insist on being run from a XGTERM terminal because that is a type of terminal that can also do simple drawings. If when you try to create a plot and all you get are lots of unreadable text it is possible that you are running IRAF from the wrong terminal — a XTERM, for instance. Type the command

```
> xgterm
```

and a new terminal will appear. If you think this terminal is too small now is the time to resize, IRAF can't deal with the XGTERM being resized while it is running. If some command spews out a lot of text into your terminal you can scroll up by pressing the Shift and Page Up buttons or down by pressing Shift and Page Down buttons. In your new XGTERM terminal type

```
> ncl
```

to start IRAF. In some places instead of `ncl` use `cl`. Which you should use depends on which version of IRAF is installed. So if IRAF is not started after you type `ncl` try to type `cl`. If IRAF starts up correctly your prompt will change to `cl>` and a list of packages will be shown. To safely quit IRAF type the command

```
cl> logout
```

at your prompt. If you find yourself having killed off IRAF in a fit on anger it is advisable to type the command `flpr` on the `cl>` prompt next time you run IRAF. This command will result in all the changes you made (IRAF could be in a messed up state) being erased. If you used `CTRL-C` during any command you even have to do one (or sometimes a couple of) `flpr`'s to get any other command running.

When IRAF starts up it first reads the `login.cl` file in your `iraf directory` for specific

settings. By editing this file you can change (or fix) some of IRAF behaviors. It is for instance possible to get the backspace key working correctly again by adding the lines[1]

```
reset editor=emacs
l_on
```

to the `login.cl` file. To make the FITS (Flexible Image Transport System) files the default file-type add the line

```
set imtype= fits
```

to your `login.cl`. IRAF can also work with `.pix` and `.imh` files. These files are a pain to work with since the header is in a different file then the header (in a FITS image they are in one file). IRAF has special commands to deal with these split up files, commands like `imcopy` and `imdelete`. This manual assumes that the default file format is set to the FITS file-type — use something else and you are on your own! To prevent IRAF from just displaying a small part of an image add this line (or edit the standard value which should already be in the `login.cl` file)

```
set stdimage=imt8192
```

which tells IRAF to assume that an image is 8192 by 8192 pixels (standard value is 512 by 512). If you do not do this, it can happen that IRAF displays just a small part of an image.
The `login.cl` file just contains all commands which are executed when IRAF starts up. This means that you can use any of the commands in the `login.cl` file that you could use on the IRAF command prompt. Another handy trick is to add the path to your working directory at the end of your `login.cl` If you are working in `/scratch/username/workingdir` just add the following line at the bottom of your `login.cl`

```
cd /scratch/username/workingdir
```

2.2 Finding your way around IRAF

The IRAF software consists of several packages that group together commands for certain tasks. The command

```
cl> ?
```

will show you the list of available packages, some of which might already be loaded. A dot at the end of a package name means that that package contains other packages. The command

```
cl> ??
```

lists all the available packages and their tasks. (However for some reason it does not list the commands of a package that is contained in another one.) All the currently loaded packages can be listed with the

```
cl> package
```

command. For packages already loaded it is possible to only list their contents by typing a

[1] There are still problems with commands that are longer than one line of the terminal, try to avoid having those long commands. See also section 2.3 for ways to run commands with lots of parameters by using `epar`.

question-mark directly in front of a package name. For example
`cl> ?noao`

lists all the commands of the `noao` package (provided that `noao` is already loaded). Loading a package is as easy as typing its name. When you successfully load a package your prompt will change from `cl>` to some abbreviation of the package you just loaded. For instance the command

`cl> noao`

will load the `noao` package and change your prompt to `no>`.

For most of the commands in IRAF there is also documentation included with IRAF. You can read this documentation using the `help` command followed by the name of the command you want help for. For instance

`cl> help splot`

will display the documentation for the `splot` command. You can scroll through the documentation using the space bar to jump a page further, the d key to scroll half a page down. To quite the help for that command push the q key. The first line of the help text. This manual has been written and tested on version IRAF V2.12EXPORT, sometimes commands change between releases. for a command describes the package it is part of. If you want to output the text of the built-in help to a file, type (e.g. for `splot`)

`cl> help splot > file_name.txt`

The built-in help pages are also useful to find out where you can find a certain command if you only know its name. The top most line of most help pages usually gives the name of the command and the packages it is in (even when these packages are not loaded). The `history` command shows you the commands you used previously in a numbered list. Assuming you want to repeat the 15th command you type

`cl> ^15`

at your prompt. However if `ncl` is set up right you can press the up arrow (several times if needed) to get the previous commands.

It is possible to interact with the UNIX system that IRAF runs on by starting a command with an exclamation mark. The command

`cl> !mozilla &`

for instance starts up the mozilla browser. When invoking programs like this it is a smart thing to put an ampersand `&` behind the command otherwise you will not be returned to your prompt. It is not always a smart thing to delete or move files from inside IRAF using the UNIX subsystem, IRAF has safe commands for that — more on which will follow.

When you start a command IRAF will let you type only a part of a command's name if that is unique — i.e. if there is no other command whose name starts with the same few letters the command will be run. Finally, to find out which version of IRAF you are running you use the `news` command

`cl> news`

2.3 Commands and their options

Commands in IRAF take parameters that can be specified in several ways. You can choose to have IRAF remember any changes you made or to have it only use them once. When you set up IRAF for your account a directory `uparm` (so bastardized spelling of user parameters) is created in your IRAF directory. This directory is where IRAF stores the values of parameters that you changed. You can list the parameters for a certain command with the `lpar` command as follows

```
cl> lpar somecommand
```

Using the `lpar` command will result in list of the parameters specified for that command. The parameters between parentheses are optional, also called hidden, these will default to preset values in case they are not specified. The parameters not between parentheses are required parameters that have to be specified each time you run the command they belong to.

To specify the parameters for a certain command you specify them at the prompt after the command in the following fashion `parameter=value`. For instance

```
cl> imheader longheader=yes
```

will set the hidden parameter `longheader` to a value of **`yes`**. The command `imheader` is run and the change to the `longheader` parameter is forgotten after just one run. When you are doing a repetitive task in IRAF (an observational astronomer's lot) it is much better to change the values of hidden parameters in a way that IRAF remembers. This can be achieved with the `epar` command. Running

```
cl> epar splot
```

changes the terminal into a sort of editor with a list of parameters similar to that of `lpar`, the difference being that the parameters can now be changed. With the up and down arrows you can move through the list of parameters. When you want to change the value of a parameter just move to that parameter and start typing in the new value, followed by enter to make the changes stick. To exit from the parameter editor back to the prompt you have several options. Typing a colon `:` allows you to give commands to `epar` just by typing one of the following commands (followed by enter):

`q` to quit `epar` without saving the changes you made.

`w` to save your changes without quitting `epar`.

`wq` to save your changes and quit `eqpar`.

`go` to quit `epar` and run the command with the changed parameters.

The command `unlearn` followed by the name of a command will let you reset the parameters to their default values.

2.4 The icfit routine

Fitting graphs is a common task in IRAF. Many commands use the `icfit` subroutine to fit them. The name `icfit` stands for "Interactive Curve fitting". In this manual, the `icfit` routine will be used a couple of times. The `apall` command uses it for background fitting and tracing the aperture. `identify` uses it to fit the relation between pixels and Ångstroms and `colbias` will use `icfit` to fit the bias to the overscan region[2].

[2]For more information about `identify` and `colbias`, see the sections about these commands.

When `icfit` is run it presents a window [3]. The `xgterm` window will become unresponsive when the graphic window is active and waiting for input. The dotted line represents the fit and the little crosses represent the data to be fit. To get some help about which keys you could press or which commands you could give in, just tap the **`?`** key. A list of all possibilities will appear in the xgterm-window. With the the **`down`** arrow key, you can scroll down a line. Pressing the **`spacebar`** will give a next screen. When you found what you are looking for, press **`q`** and you will return to the graphics screen.
Now, let's just start to look at the different options you can change in the `icfit` screen. For the fitting parameters, hit the colon `:` followed by one of the following options and a setting (e.g. `:function legendre` and then press the **`ENTER`** key.

`function` Lets you choose the type of function you want to fit. There are four functions available. The `legendre` and `chebyshev` polynomials are normal mathematical functions. The `spline1` and `spline3` are more complex mathematical objects which are piecewise fits to the data points. [4].

`order` The order of the function. For `legendre` and `chebyshev` this is clear. For the splines, this is a measure of the number of lines to be fit. A higher order function gives a better fit, but given a high enough order you could fit anything perfectly making your fit meaningless.

After changing the order and the function, you will have to hit the `f` button to refit the data with the parameters you put in. If you are able to see a dashed line through your data, just try to get it fitting more or less to the shape of it. Of course, you could use a 2000000^{th} `spline3` to fit every point and line separately, but that would always be overkill. In general you want your fit to be as easy as possible. After fitting you can see the data and the fit in some different ways by hitting some of the keys. Here a list of the most interesting features:

- **`h`** Plots the data points and the fit versus the input parameter. In our case the bias vs. the line number.
- **`j`** Plots the residuals versus the input parameter. The residual of a point (x_i, y_i) and a fit $f(x)$ is defined as $y_i - f(x_i)$. If the residuals are 0 everywhere, the fit is perfect.
- **`k`** Plots the ratio of the data to the fit. This defined as $y_i/f(x_i)$. If this ratio is 1, the fit is perfect.
- **`l`** Plots the non-linear component of the data. To get this a linear fit is made between the begin and endpoint of the data and subtracted from it.

After fitting, just tap `q`. `IRAF` will take over from here on out and will use the fitted values to do the calculation you asked for[5].

2.5 Using file lists

Sometimes, you want to do the same thing over and over again with a large number of images. For this you can use a trick which we will call a "file list". For some tasks, such a list is even compulsory. Let's assume you have 10 files in a folder and you want to subtract 8000 from all those files (on a pixel by pixel basis). First make a list of the files, which would look something like this:

```
file1.fits
file2.fits
```

[3] image still to be made
[4] the data is divided in several pieces and for each piece a polynomial is fit. The order of the polynomial is 1 (3) for the `spline1`(`spline3`) function
[5] We will come back to this when describing `apall`, `identify` and `colbias`.

```
file3.fits
file4.fits
file5.fits
file6.fits
file7.fits
file8.fits
file9.fits
file10.fits
```

If you have all files in the same folder, there is a very nice trick to make such a list. Open up a Linux console window and go to the folder your files are in

```
computer: username> cd /scratch/username/workingdir/
```

and now, just use the `ls` command and dump the output in a text file.

```
computer:/scratch/username/workingdir/ > ls *.fits > filelist.txt
```

Now make a copy of your file list

```
computer:/scratch/username/workingdir/> cp filelist.txt output.txt
```

and use your favorite editor to change the filenames in the output filenames. Normally this is done by adding a letter to them or by putting them in another folder. For the sake of the example we will show both at the same time by adding an "s" in front of the filename and putting all the file in the sub-folder "finished"

```
finished/sfile1.fits
finished/sfile2.fits
finished/sfile3.fits
finished/sfile4.fits
finished/sfile5.fits
finished/sfile6.fits
finished/sfile7.fits
finished/sfile8.fits
finished/sfile9.fits
finished/sfile10.fits
```

To use this file lists, put them just where you would put the filenames in IRAF and add an @ sign in front of them. Our example `imarith` command will then look like:

```
no> imarith @filelist.txt + 8000 @output.txt
```

So now, you will just have to type a lot less and everything is ordered in a very nice way! The chapter A about the UNIX commandline contains a section on the use of `awk`, which is also useful in this context.

Chapter 3

Images

In this chapter we will treat the the standard files which IRAF operates on, the FITS files[1]. What they contain, how you can view or plot them and what the proper way to handle them is. A FITS file can contain an image, several images or spectra. FITS files consist of one or more so-called Header Data Units. The header is a list of keywords and their corresponding values, typically stuff like the time at which the image was acquired, the exposure time, the instrument that created it and the coordinates on the sky.

The installation of IRAF and `ds9` (an image viewer) is a bit of a black art that depends on the underlying operating system and requires special privileges we will not describe the process of installing IRAF here. We simply assume IRAF and `ds9` are correctly set up — for the situation at the "Anton Pannekoek Instituut" there is some information in the Appendix **??**.

3.1 Viewing one dimensional images

For one dimensional images you can use the graphing facilities built into IRAF. The `implot` command lets you make plots of one dimensional images. One dimensional images are typically spectra and their ilk (e.g. see figure 3.1). The command

```
cl> implot blah.fits
```

will open the file `blah.fits` and display it. While the plot window is active you cannot use the IRAF command line, to use it again you will have to first press the **q** key. Doing this deactivates the window, you are then free to close the plot window. Were you not to press **q** before closing IRAF will hang or crash. If the image is two dimensional you can use `implot` to look at a cross-section of the image. The command

```
cl> implot blah.fits[*,30]
```

will show a cross-section along the x-axis (a line) in the image at a value of y of 30, likewise

```
cl> implot blah.fits[30,*]
```

will display a coruscation along the y-axis (a column). Note that this will fail if you put any extra spaces in the **blah.fits[*,30]** part of the command, you will get an error like

```
ERROR: may not convert string to other types
```

[1]This manual assumes you are working with FITS files and not the older `.imh` and `.pix` format. The latter are a pain to work with since the header and picture data are in different files. See section 2.1 for information on how to set FITS files as the standard filetype.

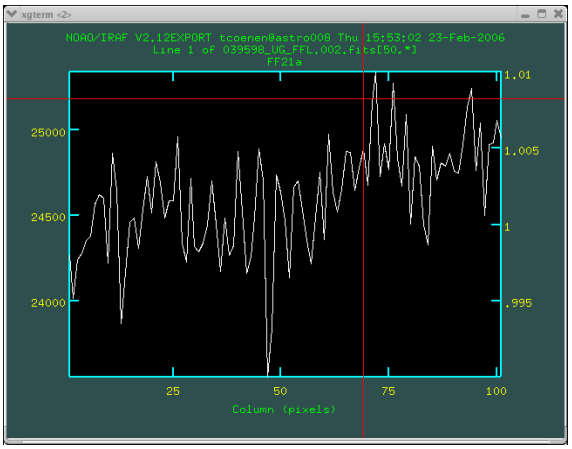

Figure 3.1: A standard `implot` view.

. IRAF can save the output of plotting routines to Encapsulated Postscript files (`.eps` files). You first need an active plot window. Press the colon **`:`** key followed by **`.snap eps`** and finally press the return key to create a plot centered in the `.eps` image. In stead of **`eps`** it is also possible to use **`epsh`** or **`epsl`** to respectively create potrait or landscaped plots.

3.2 Viewing two dimensional images

While technically not a part of IRAF the `ds9` image viewer is the way to look at FITS files. Unlike IRAF, `ds9` does have a graphical user interface which makes it a joy to use (comparatively). You can start `ds9` with the command

```
cl> !ds9&
```

You can use `ds9` by itself or in linked to IRAF. Through the menu system of `ds9` you can open images. If upon opening a fits file you don't see anything you should adjust the contrast. Press the right mouse button and hold it while the mouse pointer is over the picture and move the pointer left, right, up or down. Moving up will decrease the contrast, moving down will increase it. Moving left or right changes which photon count values are in the gray range. The bar below the picture shows a gradient to indicate what the current contrast range is on the picture.

Some times it is useful to change the scaling on the image, `ds9` has a scale button available that lets you choose between different scalings. It is probably defaulting to a linear scaling. When in a image there are both bright and faint objects the logarithmic scaling will be able to show both. A linear scaling might hide the faint objects in the lowest bin while a logarithmic scaling will show details over many orders of magnitude in brightness — just like an astronomers' favorite the log-log plot shows details over many orders of magnitude. If you use `ds9` stand alone the scale button will be available. If run through the `display` command (about which more will follow below) the scaling can be set using the `display` command's parameters. It is possible to zoom in the image using the zoom button, you can then move around the image by dragging the little green box in the smaller picture display to the left. You can also move around the image by clicking and holding the middle mouse button down and moving the mouse around. The `ds9` program also contains an option to view the header of a FITS file[2].

The `display` command is the preferred way of using `ds9` in conjunction with IRAF. To use

[2]There is also another way to get the header of a FITS file, it is described in section 2.3.

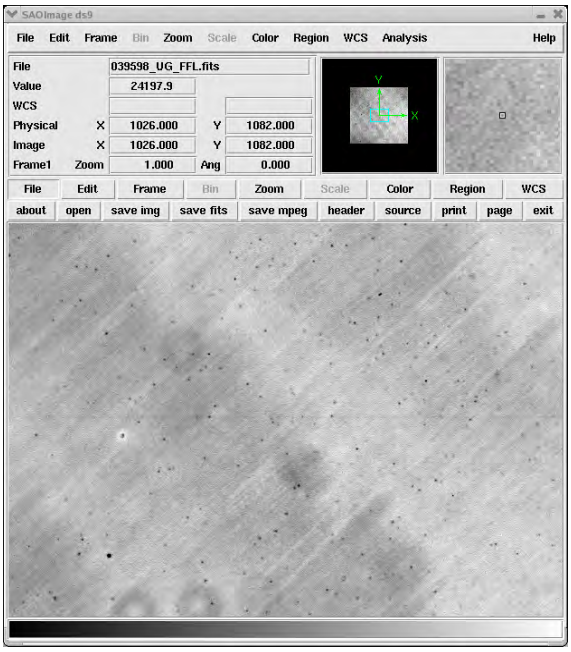

Figure 3.2: A typical `ds9` session showing a flatfield.

the `display` command IRAF and `ds9` must be set up correctly. When an image is displayed through the `display` command of IRAF not all the information in the headers of that image will be available for inspection by `ds9` (since IRAF changes the headers of files that it sends to `ds9`).

3.2.1 Getting at the statistics of an image

When looking at an image in `ds9` moving the cursor around will show the location of the pointer and the value of the image at that spot — or that is what one would hope. The statistics that `ds9` reports can be off quite a bit. To get the actual statistics in regions of an image there are several commands. For a quick and general look at the statistics of a whole image one uses the `imstat` command. For more detailed inspection the `imexamine` is available. For `imexamine` to work IRAF and `ds9` must be set up correctly. The following text assumes that the parameters for `imexamine` are not changed from their default values (see section 3.2.2 for the options of `imexamine`).

First you need a display where you can select regions of interest in some image so start `ds9`;

```
cl> !ds9&
```

You will now see an instance of `ds9` appear. To examine an image run the `imexamine` command by typing

```
cl> imexamine image.fits
```

This will open the image **`image.fits`** in `ds9`, change your cursor to a blinking circular cursor and move it to the `ds9` window. You can now select regions for inspection by focusing[3] the `ds9` window. Now move your cursor to an interesting spot and press one of the following keys:

[3]Focusing is jargon we will use throughout this manual to describe the act of making a window active (accept input). This is usually done by clicking on the bar on top of the window — focusing is not something that IRAF controls. Some strangely configured computer may focus a window when you hover over it. This annoying behavior is called focus follows mouse.

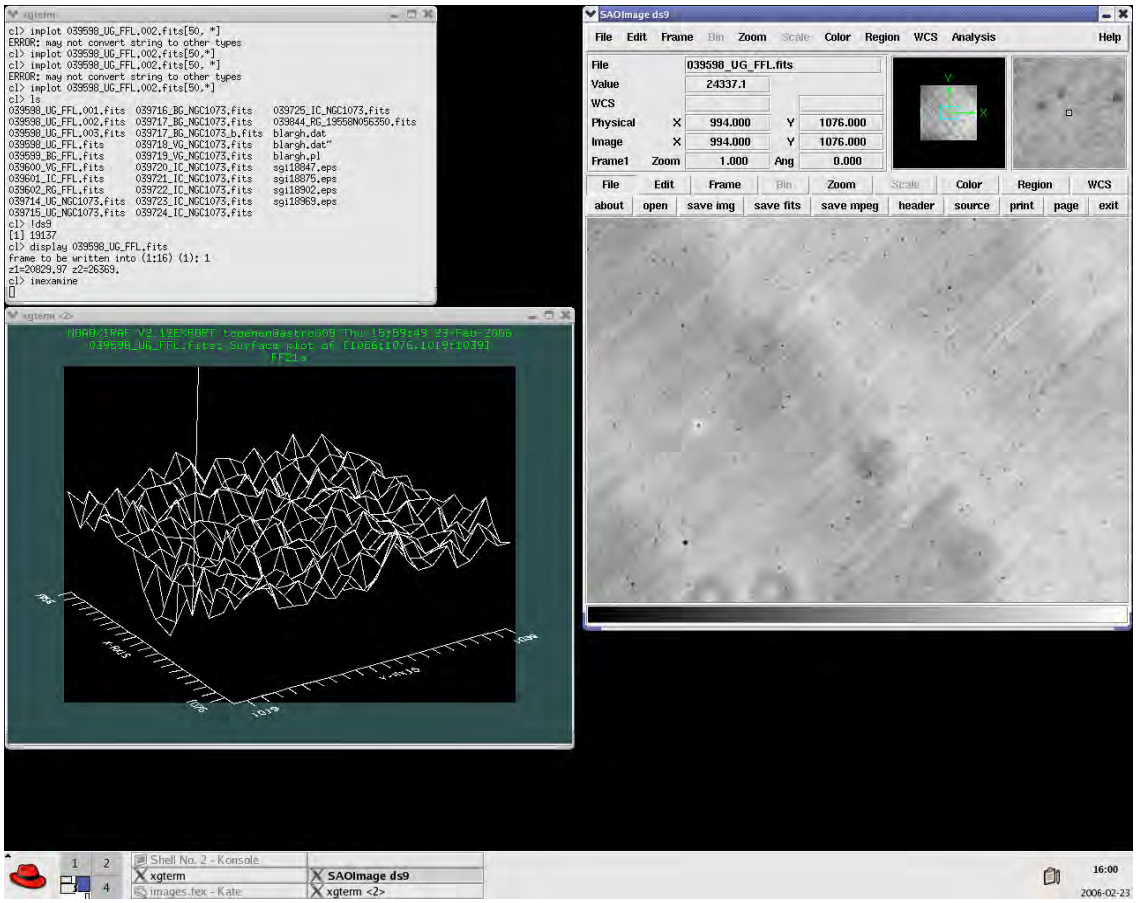

Figure 3.3: Using `imexamine` to examine a flatfield, the plot window shows a surface plot of a region in the flatfield.

q This quits the `imexamine` program. Pressing **q** will only work if and when the `ds9` window has focus. The cursor will change back into a normal cursor and you will be dropped back to the IRAF prompt. After this happened it is save to close any extra window that `imexamine` opened.

c Pressing **c** will result in a graph of the current column. You can also use it to plot a graph of the average of several columns, see 3.2.2.

l Results in a graph of the current line or average of several lines if you have changed the relevant parameters, see 3.2.2.

s This creates a surface plot of the region around the current cursor position.

e This creates a contour plot of the region around the current cursor position.

h This creates a histogram of the region around the current cursor position. That is it takes all the pixel values of the pixels in the region around the cursor and creates a histogram out of these values.

o The **o** is short for overplot, this allows you to plot another graph on top of the last graph. After pressing **o** the last graph is retained and the next graph (press one of the graphing keys) is drawn on top of it.

j Perform a one dimensional Gaussian fit to the current lines.

k Perform a one dimensional Gaussian fit to the current rows.

x This prints the current and the value of the pixel that the cursor hovers over. These values will be printed in the IRAF window.

z Prints the pixel values of a 10 by 10 grid around the current position. These values will be printed in the IRAF window.

If a command seems to be ignored check whether the right window has focus, for the commands mentioned above that would be the `ds9` window. Imexamine can also write output to files with the following commands;

t Press the **t** key to output an image centered around the cursor position with a size and name controlled by parameters of the `imexamine` command. For relevant values see section 3.2.2. IRAF will show the name of the file that it creates.

w Pressing the **w** switches `imexamine` between writing output to the terminal running IRAF and a log file specified in the parameter file of `imexamine`. See subsection 3.2.2 for the relevant options.

Although we've described quite a few of `imexamine`'s capabilities it can do a lot more. We refer you to the help files contained in IRAF for all the options.

Finally it is possible to switch between the so called input cursor (the blinking cursor hovering over the image in `ds9`) or the graphical cursor (the red cross in plot windows). To switch to the graphical cursor press **g** whilst the hovering somewhere in the focused `ds9` window. The cursor will now jump to the plot window and change in a red cross. To change back again press the **i** key whilst hovering over the focused plot window. If all is well the cursor will jump back to the `ds9` window and change back into the input cursor. Switching to the graphics cursor is useful to create `.eps` files of plots you create using `imexamine`. Just press **g** to jump to the plot window then press **:** followed by **.snap eps** and a press of the return key. It is important to keep in mind that you can only quit the `imexamine` command when you are in the input cursor mode and the `ds9` window is also focused. If you forget this you might not be able to resume your IRAF session.

3.2.2 The parameters of `imexamine`

In the last subsection we described the behavior of `imexamine` in its default settings. The `imexamine` command is special in the sense that some of its options are spread over a few hidden commands. The parameters for the column plots are for instance contained in the parameter file for the (hidden) `cimexam` command. But lets first get into the normal parameters contained in the parameter file of the `imexamine` command.

Here is a list of some of the parameters for the `imexamine` command;

`output` The output parameter can be set to the root name of images that are created using `imexamine`'s **`t`** command. If no root-name is specified the name of the image that is being examined is used as a root-name. The `imexamine` command then appends a number like `.001` to the root-name to create unique filenames for the newly created images.

`logfile` Self explanatory really, this parameter contains the name of the logfile that receives the output of `imexamine`.

`ncoutput` This parameters controls the number of columns (the horizontal size) of an image output by using the **`t`** command from within `imexamine`.

`nloutput` The same as `ncoutput` but for the number of lines — that is the vertical size — of the output image.

The behavior of the **`c`** command is controlled through the parameters of `cimexam`, those of **`l`** through `limexam` etc. In all the following hidden commands contain options relevant to `imexamine`: `cimexam`, `eimexam`, `himexam`, `jimexam`, `kimexam`, `limexam`, `limexam`, `rimexam`, `simexam` and `vimexam`. They share quite a few parameters. We will describe some of the more important parameters (shared or not).

`angh` This parameter controls how the a surface plot is rotated around an axis perpendicular to the examined image. Contained in the parameter file for the hidden `simexam` command.

`angv` This parameter controls the how steep the surface plots created with `imexamine` are. A value of 90 is a top view, a value of 0 is a side view. Contained in the parameter file for the hidden `simexam` command.

`banner` This parameter controls whether there is a banner containing IRAF user name, time, title and image name over the plots you create using `imexamine`. Contained in all the aforementioned hidden commands.

`ncolumns` The `ncolumns` parameter controls the number of columns used to create surface plots, contour plots and histograms. Contained in the parameter files for the commands `eimexam`, `himexam` and `simexam`.

`nlines` The `nlines` parameter controls the number of lines used to create surface plots, contour plots and histograms. Contained in the parameter files for the commands `eimexam`, `himexam` and `simexam`.

3.3 Working with image files and regions of images

Although it would be perfectly possible to use the UNIX shell to move around files, delete and copy them, it is not the recommended way of working in IRAF. IRAF provides some commands of its own to do these thing that at times are safer to use. Since fits files can point to each other when you move them around you have to update them, IRAF will do this for you provided that you use the commands built into IRAF. So instead of typing something like

```
cl> !rm somefile.fits
```

you can use

```
cl> delete somefile.fits
```

to delete the file `somefile.fits`. The `imcopy` command can copy images or even just regions of images. The command

```
cl> imcopy imageA.fits[10:20,40:50] fileb.fits
```

will copy a region from `imageA.fits` and copy it into the new file `fileB.fits`, leaving the part between square brackets of will result in a straight copy of `fileA.fits` to `fileB.fits`. You can also use imcopy to flip images, typing

```
cl> imcopy fileA.fits[*,-*] fileB.fits
```

results in flipping the image `fileA.fits` over the x-axis. If you replace `fileB.fits` with `fileA.fits` it will just flip the image without copying.

3.3.1 Looking through the header of an image

The headers of FITS files are available in IRAF through the `imhead` command. To use it simply type

```
cl> imhead filename.fits
```

. If this results in a line like this

```
039598_UG_FFL.fits[2158,2044][ushort]:  FF21a
```

the command is not set to show the full header. Change the `longheader` parameter of the `imhead` command to the value **`yes`** instead of **`no`**. Running `imhead` with `longheader` set to **`yes`** results in a lot of text scrolling through the terminal. It is handy to output the header to a text file that you can then read with a text editor. To do this type

```
cl> imhead filename.fits > sometext.txt
```

the header of `filename.fits` is now output to the file `sometext.txt`.

3.4 Simple arithmetic on images

When working with images in astronomy, you will often want to subtract, divide or multiply images with each other — or divide or multiply with some fixed numerical value. IRAF has facilities built to achieve this. The best way to do this is with the `imarith` command. The syntax of this command is highly intuitive. Let's give an example input

```
no> imarith image1 + image2 image3
```

Here, we add up image2 to image1 and save the result in image3. To divide image1 by image2, just replace the + by a / and you're good to go[4]. For multiplication just use * and for subtraction a -. You can also replace a file name with a number. This will act as if an image

[4]Do remember that IRAF will not overwrite a file when you're using `imarith`. It will spew out an error message if the file you wanted to write to already exists

would be used with the same value on any pixel, so

```
no> imarith image1 + 8000 image2
```

will add 8000 to each pixel value of `image1` and save the result in `image2`.
Another trick you will have to do a lot is to add up different observations of one object to get a better Signal-to-noise ratio. For this, the `imcombine` task can best be used. When combining files, you will have to choose between the mean and median filter. A good rule of thumb is to use the mean filter whenever there are less than 5 images and the median when there are 5 or more images. To use the mean filter just type

```
no> imcombine.combine="average"
```

For the median filter type

```
no> imcombine.combine="median"
```

before you use the `imcombine` command. The syntax of the `imcombine` command is as follows; First **`imcombine`**, then the images you want to combine, separated by commas (only commas, no spaces!) and the name of the output image. So to combine four images, called `img1`, `img2`, `img3` and `img4` and saving the result to an image called `addup.fits` just use the following command;

```
no> imcombine img1,img2,img3,img4 addup.fits
```

Chapter 4

CCDs and basic image clean up

4.1 The CCD

The CCD was invented in the 1960's at Bell Labs to serve as computer memory. Nowadays they are used as image sensors, not as memory. CCD is an acronym for Charge-Coupled Device. CCDs in astronomical instruments have a light sensitive grid. Some applications of CCDs like scanners might have just one line of light sensitive elements but in astrophysics you would be wasting a lot of expensive photons that way. When photons strike the light sensitive area of a CCD some electrons are released within it, those get caught in one of the potential wells within the grid. The potential wells are formed by applying voltages across electrodes that are on top of the chip (or on the undersides sometimes), a few electrodes are used for each line in the grid. By cycling the voltages across these electrodes you can shift the potential wells and their contents one line at a time (but with all the lines moving in lockstep, this is what the name Charge-Coupled Device refers to). When a line reaches the edge of the chip the pixels in that line are read out, the resulting voltages amplified and then converted to digital signals.

The counts caused by thermally excited electrons are called the dark current of a CCD, they will also be present if a CCD is read out without having been exposed to light. Images like this will be called dark-frames throughout this manual. The fact that dark current accumulates even when a CCD is not exposed to light, can be exploited to get rid of the dark current in the images taken by the CCD — just subtract a dark-frame made under the same circumstances as your image from your image. We will sometimes refer to images that contain the science targets as science-frames. The electronics in a CCD also add some base voltage into an image to make sure that the voltage amplification step is successful, this base voltage is called the bias voltage. A CCD read out without exposing it to light and with an exposure time of zero seconds yields a bias-frame also called a zero-frame. A dark-frame also contains the bias. There are a few ways of correcting for dark-current and bias, the practical steps that you will have to go through to correct for them will be dealt with later.

A CCD might also not be equally sensitive to photons across its entire light collecting area. Some pixels might be more sensitive than others, one will also have to correct for this effect to be able to use the data from the CCD. By taking a picture of some spatially homogeneous light source, one that covers the entire light sensitive area of the CCD, you get a so called flat-field. Since every pixel had the same amount of photons fall onto it you can correct for local changes in sensitivity by dividing the image through the flat-field. Flat-fields also correct for uneven illumination of the CCD. For spectrographic applications the way you perform the flat-field correction is different, again the practicalities will be dealt with in due course.

To create good flatfields and darkframes you must combine several of them into one flatfield. By combining several flats you reduce the amount of noise in the combined flatfields and darkframes. Combined flatfields can be created out of the flatfields taken over several days (typically you only get two flatfields per filter per night from a telescope one from dusk and one from dawn). You

have to check whether the flatfields look the same before combining them though! If you don't check for differences you may well end up "correcting" images for problems that were not present when they were taken.

CCDs have a range of energies of photons that can excite electrons. Some photons free up more electrons than others. Since a CCD only counts electrons you lose any information about the energy of the incoming photons. In applications like (consumer) cameras the CCD can be made color sensitive by adding a grid of colored masks to pixels (lowering the resolution and sensitivity of the CCD) or by having a prism split the incoming light and using several CCDs for the different color channels (like the more expensive video cameras do). In astronomy typically several exposures are made using different filters, which can then be combined into color images. These color images are usually false colors since the filter's transmission curves don't correspond to the transmission curves of our eye's cones.

When CCDs are overexposed the potential wells fill up and the electrons start spilling over into neighboring potential wells — this process is called blooming. Blooming can be stopped by adding extra electrodes to the chip that drain any excess electrons. Doing this will take away from the light collecting area of the CCD so it is a trade-off. Something else to keep in mind is that CCDs are typically not linear in their sensitivity to electrons once their potential wells start getting fuller. It might be that as the wells fill up not as many electrons get caught per new incoming photon. To deal with this it is useful to find out over which range the CCD is linear and not expose it to more light, you might not be able to use the full range of the analog to digital converter.

Cosmic ray hits are also a problem for CCDs, as they were for observers that used photographic plates. If you have several exposures of the same object you might be able to take the median of several images to get rid of them. For flat-fields, bias-frames and dark-frames you always take a few to get rid of cosmic ray hits — but also to make sure you don't add extra counting noise to your pictures.

You might come across the term ADU, Analog-to-Digital Unit , this refers to the amount of electrons that make up one count in the final image. The value of the ADU depend on the amplifier that is used.

Dark-frames can either be scalable or non-scalable (We refer to the latter as normal). The normal dark-frames are made with the same 'exposure' time as science-frames that you correct with them. A normal dark-frame is made with the same exposure time as the science-frame that you want to correct for dark current. The darkframe should be created under the same circumstances as the science-frame ideally right before or after the science-frame.

These non-scalable dark frames might still have the bias since subtracting them from your science-frames and flat fields gets rid of the dark-current and bias in one go. Scalable dark-frames are dark-frames that can be scaled to different exposure times of your science frames. For this to work you must subtract the bias from you raw dark-frames, since the bias does not scale with exposure time.

Ideally one would like to know the properties of the CCD for each and every moment that observations are performed. Since time at an observatory is precious shortcuts are taken. Flatfields are usually made at the beginning and the end of the night — in the case of so-called sky-flats of the twilight sky. Flats created using artificial light are also created at moments that one isn't able to perform actual astronomical observations. The underlying assumption is that the optical properties of the telescope and differences in sensitivity across the CCD don't change very quickly. This is an assumption that can be checked by comparing flats made at different moments to see whether they still show the same structure — if they do there is no problem. The dark current and bias are not as stable so one must measure them more frequently. It is possible to approximate the bias structure, dark current and problems during read out by exposing only part of the CCD and then reading the whole thing out. The unexposed part of the CCD — the so-called overscan region — contains roughly the same dark current and bias signal as the exposed part. Now it is possible to correct for dark current, bias and read out problems for each image taken without having to make dark-frames after each exposure. The Merope imager at the Mercator observatory contains a T210 CCD that has an overscan region.

4.2 Example: cleaning up Merope CCD data

Although the steps one has to go through to clean up the raw data from the Merope CCD (the CCD at the Mercator observatory on La Palma) are the same there are several packages one could use. This chapter is section is based on the 2005 configuration of the Merope CCD. Personally I went through the following steps. First I subtracted the bias and dark-current from each of the images (including the flatfields) using the overscan region. I then fixed bad pixel data using a mask. I normalized all the flatfields to an average value of 1. Then I combined the flatfields into one flatfield per filter per day. You might want to combine all available flatfields for a number of days into one flatfield if the flatfields all look alike — and only if they look alike[1]. Then I divided all the science-frames through the normalized and combined flatfields. Then I combined the science-frames into one image per filter.

4.2.1 Bias and dark-current subtraction using the overscan region

The Merope CCD has two overscan regions that are used to deal with bias and dark current. These regions lie in two columns around the image area. IRAF contains commands that deal with overscan regions (aside form the `ccdproc` functions that will be described later) they are `colbias` and `linebias`. These commands are part of the `imred` package that itself is contained in the `noao` package.

```
cl> noao

no> imred
```

Before you can run `colbias` you have to set some parameters to the appropriate values. Edit the parameters of `colbias` using `epar`.

```
im> epar colbias
```

Set `trim` to the image (the part containing the data, not including overscan regions), for Merope the appropriate value is **`[49:2095, *]`** . Set `bias` to the overscan region, for Merope that would be **`[2096:2158, *]`** — note that this uses only one of the two columns of the overscan region. The `interac` parameter can be set to either **`yes`** or **`no`** depending on whether you would like the task interactively. Only set it to **`no`** after having chosen an appropriate function to fit see section 2.4. Now run `colbias` by typing the following command

```
bi> colbias inputfile.fits outputfile.fits
```

to subtract the bias and dark current from **`inputfile.fits`**, cut the overscan regions of of **`inputfile.fits`** and write the results to the file **`outputfile.fits`**.

4.2.2 Performing the flatfield correction

Assuming you have corrected both the flatfields and science frames for bias and dark current you can move on to the flatfield correction. I performed the flatfield correction using the `imarith` command. I first normalized all the flatfields. To do so you first need the mean value of the pixels in your frame. Run `imstat` on all your flatfields to find these values

```
cl> imstat flat.fits
```

this will output among other things the mean value of the flatfield. Then use the command

[1] This will reduce the noise in the flatfield see the previous section.

```
cl> imarith flat.fits / 10 flat_n.fits
```

to divide the image flat.fits through 10 write the results in the file flat_n.fits. You of course have to replace the **10** in this example by some appropriate value for the mean of the image. Then combine several flatfields (if available) into one flatfield using the `imcombine` command. Before you combine any flatfields it is important to check whether they show the same structure. If they do it is save to combine several flatfields to get better statistics. If they don't however, you will end up with a flatfield that is some mix of appropriate and inappropriate flatfields. If you use a flatfield like that you will ruin you images! If you have more than a few appropirate flatfields set the `combine` parameter to **median** otherwise leave it at **average**. Now run `imcombine`

```
cl> imcombine flat1_n.fits,flat2_n.fits,flat3_n.fits masterflat.fits
```

to combine images flat1_n.fits, flat2_n.fits, flat3_n.fits into the image masterflat.fits. Now that you have an appropriate flatfield, you divide your science-frame through your flatfield.

```
cl> imarith scienceframe.fits / masterflat.fits output.fits
```

The flatfields are normalized to make sure that you don not lose to much numerical precision[2] when you divide a science-frame with low counts through the flatfield.

4.2.3 Masking bad pixels and using `fixpix` and `mskexpr`

IRAF has a built-in command that will "fix" bad pixels in an image. The `fixpix` command can be used to replace bad data (bad pixels in a CCD show up on every image that is made with it) by an interpolation between known good data around the bad pixels. Note that the images are overwritten in this process! (Be sure to make backups of the original files.) These bad pixels are defined in a mask. I used the `mskexpr` command to create a bad pixel mask. Although I will only describe the way one creates a mask that takes out bad pixels `mskexpr` can do much more complicated things (just look at its help file if you're curious). To specify a bad region first find out the coordinates of the bad pixels using for instance `ds9`. Create a text file with for each bad region a line like the following

```
box(x1, y1, x2, y2) ?  1 :  0
```

where $(x1, y1)$ is the upper left and $(x2, y2)$ is the lower right hand corner (assuming you did not flip or rotate any axis) of the image. The last part of the line `? 1 : 0` means that while inside the box the pixels (of the mask) will be set to 1 and outside of it to 0. There are more geometrical functions supported by `mskexpr`, they are listed in its help file. When trying to ascertain the positions of image defects from within `ds9` keep in mind that displaying images through IRAF might change the physical pixel position that is displayed (the cursor position). Open files that you want to correct directly in `ds9` using its menus. Save the text file and set the parameters for `mskexpr`

```
cl> epar mskexpr
```

. Set the `dims` parameter to the appropriate image dimensions (you can find the dimensions of an image in the headers of the `.fits` files). Run `mskexpr` using the following command

```
cl> mskexpr @filename.dat mask.pl
```

Here **`filename.dat`** is the name of the file you created that contains the bad regions and **`mask.pl`**

[2]The numerical precision has to do with the way that computers internally represent floating point numbers.

is the mask that `mskepr` creates for you. Now that you have a bad pixel mask it is time to run `fixpix` on the images that need correcting. Create a text file with all the images that you want corrected and run the following command;

```
cl> fixpix @list.txt mask.pl
```

If all goes well you will be left with corrected images. You could create your own bad pixel mask as described here but observatories quite often have bad pixel masks available for their CCDs[3].

The `fixpix` command will overwrite the bad pixels with a linear interpolation between good pixels around the bad ones. This is fine from a cosmetic point of view, but if the object of interest is where there are bad pixels you cannot really trust that data even after "fixing" it — so be careful.

4.3 The `ccdproc` command and the cleaning of images

To be able to get reliable data from a CCD one must correct for the above mentioned effects. One can correct by hand for bias, dark currents, pixel-to-pixel changes in sensitivity but IRAF also provides commands to do these things semi automatic using the `ccdproc` command which is part of the `imred` package. If you work on image mosaics you should use the `mscred` package. The `mscred` package is similar to the `imred` package but has to take into account the ways that CCDs might influence each other.

Now, let's fist start the correct package (`ccdred`),

```
cl> imred
```

followed by

```
im> ccdred
```

and edit the parameters of `ccdproc`

```
cc> epar ccdproc
```

Now let's look at the parameters before continuing. The **`images`** option specifies the input image (or list) name (this will be asked when you run the command anyway). The **`output`** specifies how the corrected image(s) should be called. Just empty the contents of the **`ccdtype`** parameter (make it `""`)[4]. Then you will have a list of tasks which you might want to execute. If you would like to use a certain task, change its value to `yes` and if you do not want to use it, the value should —very surprisingly— be `no`. See table 4.3 for a list of the most commonly used tasks and what they do.

Now you will see a lot of parameters for all the tasks. Their order is more or less the same as the order they have in the listing. We will just assume that you set the appropriate values to `yes`. First thing to look at is the **`readaxi`** parameter. This sets **a lot a lot a lot a lot**. When you want to use a bad pixel mask, you should specify a pixel list (a file, probably with the .pl extension). `biassec` is the overscan region and `trimsec` is the section which will yield the final image (use this when using the overscan for finding the bias; see also section4.2.1). You will also have to set the `functio` and `order` variables to the type of fit function and function order you would like to use for fitting the overscan. `zero` and `dark` are the filenames of the biasframe and darkframe the

[3]Usually somewhere on their websites.

[4]Some telescopes assign a special tag to the fits file when making it. This tag is different for each type of file (flatfield, image, bias, etc.) This is used to automatically select the right images for the chosen action when processing a lot of images. For us this is however too cumbersome.

taskname	Task description
`fixpix`	Fix bad pixels using a bad pixel map (see section 4.2.3).
`oversca`	Use an overscan region to fit bias (see section 4.2.1).
`trim`	When using prescan and overscan, this trims the "image" area. Normally used in combination with `oversca` (see section 4.2.1).
`zerocor`	If you have a separate bias frame, use this. **ref?**
`darkcor`	If you have a separate darkframe, use this. **ref?**
`flatcor`	Add up flatfields, correct them for bias and dark current (if used), normalize them and subtract the result from the (also bias and dark corrected) image. (see section 4.2.2)

program should use if you want to use separate darkframes and zero frames. The `flat` variable takes the filename(s) (you can use *) of the flat field(s) you would like to use for division.

Chapter 5

DSS gauging and shifting of images

Probably, you will have made different exposures of one object to get a better signal to noise ratio. There is also a high probability that the object will not be on exactly the same place on your image. To reduce the noise, it is even better to have some shifted images. To be able to add up the images, they all have to be shifted in such a way that the stars are on top of eachother. We will do this using an external package which makes it far more easy. Because you need an arbitrary coördinate system to "pin" on all your files, it is advised to take an image from a Digitized Sky Survey (DSS) which has as an advantage that you will also be able to find absolute coördinates on your image. You can find images which are good enough on the website of the ESO Online Digitized Sky Survey [1]. There you can put in the coordinates and field radius of the reference image you want and download a FITS file. For this example we will assume that this file is called `dss.fits` and that the target image is called `image.fits`. In this example both files are located in the`/scratch/username/workingdir` folder.
For this procedure we will use the program **koords** which is part of the `karma sutra` system. before you are able to run the program, you will have to type the following command in a linux console

```
computer:/scratch/username/workingdir/> source /home/sw-astro/karma/.login
```

Now, change your directory to the one in which your files are. In this folder type `koords &` and two windows will pop up. One very important thing to be noted is that you should **never** close any screen by pressing the little cross in the upper right corner because this will always kill the whole program and you will have to do everything over again. One of the windows contains a button **Ref. Image**, click it. Click on the reference filename (in our case `dss.fits`). Then click on the **Target Image** button in the other screen and click on the target image (**image.fits**; see figure 5.3). Probably you will not have an ideal view on the reference image while the dss image is rather clear. For this, click on the **Intensity** menu of the Target Image screen and select **IScale for Dataset**. You will now see a histogram. Click on the **Zero Left** button. Then right click somewhere near the right edge of the peak at the far left side. then click the `Zoom` button and you will be able to see the intensity peak very clearly. Move the right side of the box by clicking the right mouse button on the graph. You can move the left edge of the box by clicking the left mouse button. Put the box in such a way that the whole peak is inside of it (like the peak in figure 5.2). In the background you will see the intensity change. When you can see the stars clearly click **close** (not the little cross!). Now you will have to tell the program which stars are the same on each frame. Do this by middle-clicking on a star in the reference frame and middle-clicking the corresponding star in the target frame. A circle with a number next to it will been drawn around both stars. Do this for a couple of stars (you will get somethin like figure **??**). Now click on the

[1] http://archive.eso.org/dss/dss

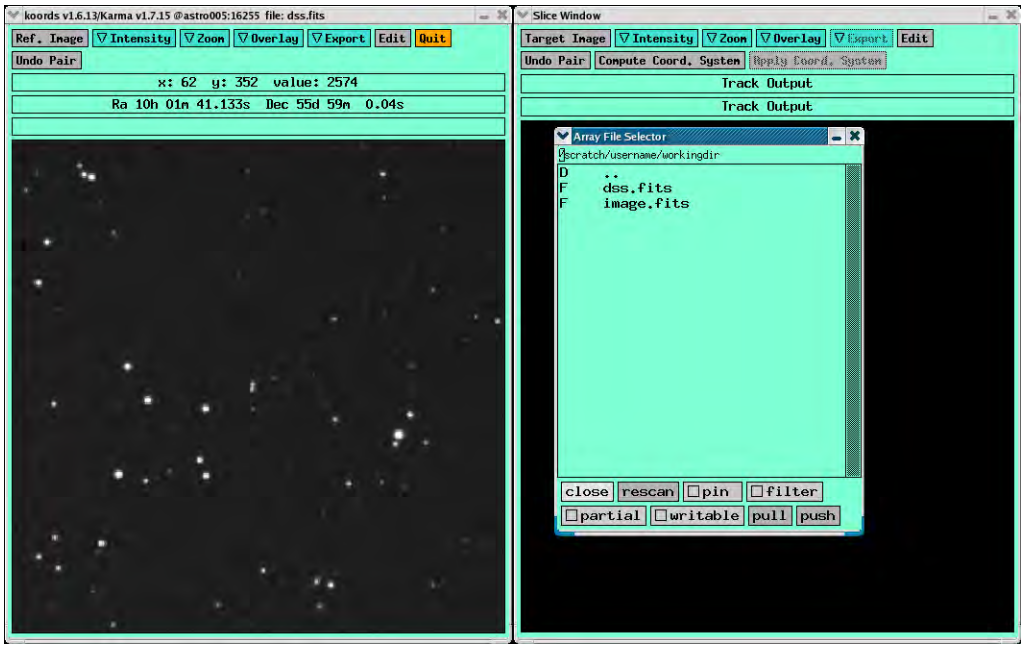

Figure 5.1: **Left:** The reference image has been opened. **Right:** The selection screen for the target image.

button **Compute Coord. System**. This will give some output in the shell (the one in which you starded koords) which looks like this

```
#  Ra              Dec              Tx     Ty      PDx    PDy     Dx     Dy
wcs_astro_transform: no LMtoXY matrix
0  10h 01m 40.950s 55d 52m 12.39s   494.5  78.9    -343.4 -21.8   0.7    1.2
1  10h 01m 27.886s 55d 53m 27.31s   877.7  645.0   -731.3 -589.5  -4.0   -0.4
2  10h 01m  0.151s 55d 52m 41.65s   651.3  1854.7  -498.2 -1796.0 2.8    2.8
3  10h 01m 38.843s 55d 54m 23.30s   1171.2 168.8   -1020.1-112.9  0.7    -0.1
4  10h 01m 36.180s 55d 53m 41.01s   949.7  282.9   -801.6 -228.9  -2.3   -1.9
5  10h 01m 29.062s 55d 52m 24.92s   558.6  593.9   -408.8 -538.9  -0.5   -0.8
6  10h 01m 28.313s 55d 52m  5.33s   458.8  627.8   -307.6 -571.6  0.8    0.3
RMS pixel error: 0.3

Header information:
Projection: ARC (rectangular)
Reference:  Ra 10h 01m 19.217s  Dec 55d 53m 54.63s   pixel: 1024.5  1023.0
Co-ordinate increment:  Ra 0.193  Dec 0.193  (arcsec/pixel)
Rotation: 90.116 degrees
```

Now click on **Apply Cord. System**. Click on the **Export** menu and select the **FITS (whole dataset)** option. The standard name of the file will be the name of your old file with an extra **.fits** extension (in our case **image.fits.fits**). Now the image will be callibrated with a coördinate system. When ready, let's go back to IRAF. Suppose you have three images which you callibrated in the above mentioned way and want to add them up. To do this, you will have to change a parameter of the **imcombine** command.

```
cl> imcombine.offsets="wcs"
```

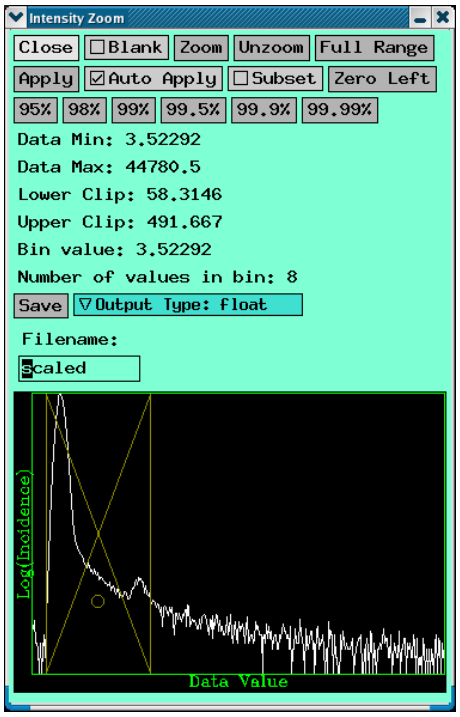

Figure 5.2: The two koords windows.

Which tells iraf to use the implemented World Coördinate Sysytem (WCS) as the reference for all images. Now you can combine `image1.fits`, **`image3.fits`** and `image3.fits` in a new image called **`imcom.fits`**.

```
cl> imcombine image1,image2,image3 imcom.fits
```

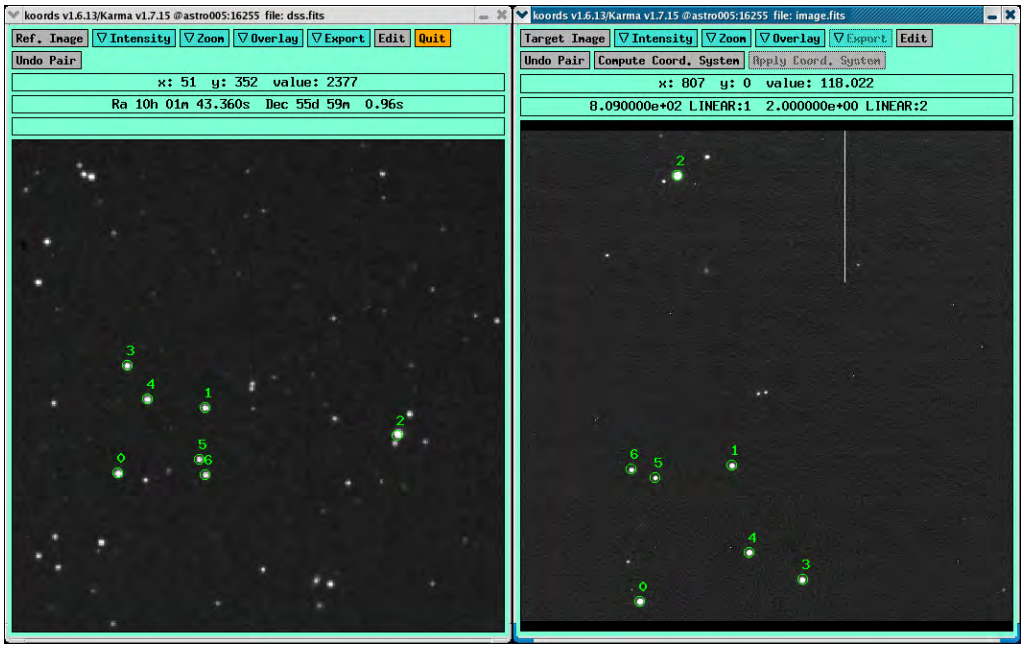

Figure 5.3: Some stars have been correllated to eachother in this image. We are ready to compute the coordinate system!

Chapter 6

Doing photometry

The goal of photometry is to find the flux (and so the magnitude) of an object in a certain colour filter. Photometry can been used to make a rough estimate ofr a spectrum when a spectrograph is not available. Another use for photometry is to detect changes in colour (the magnitude difference between two filters) or to detect any kind of magnitude changes (e.g. transient planets, gravitational lensing by non-luminous objects or object variability). Photometry can also be used to define the redshift of an object with the position of the Ly-α break. The method we will use for photometry is called aperture photometry. This method consists of picking an aperture In this case, an aperture is defined as a circle around an object on the science frame. For stars, a circular aperture does the trick[1] The number of counts from this aperture is subtracted from the number of counts from another aperture around it which is called the sky. See figure 6.1 for a schematic drawing of the apertures. To find the magnitude of the object, the number of counts in the sky

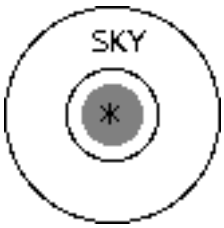

Figure 6.1: Schematic figure of a photometry aperture. The inner aperture has to include the star (*). The outer (sky) aperture is used to normalise the number of counts in the star to the background sky.

is subtracted from the number of counts in the star (normalised to the surface difference between the two).

$$\mathcal{F} = \mathrm{N}_{ap} - \langle \mathrm{N}_{sky} \rangle \times A_{app} \tag{6.1}$$

In this formula, N_{app} and $\langle \mathrm{N}_{sky} \rangle$ are respectively the number of counts in the aperture and the mean value of the number of counts in the sky (meaned over all pixels within your sky aperture). Note that the mean value of the sky is not essentially the same as the total mean value. A_{ap} is the surface of the aperture. The magnitude of a star is then defined as

$$M_* = M_0 - 2.5 *^{10} \log \mathcal{F} + 2.5 *^{10} \log \mathrm{T}_{exp} + \kappa f_z \tag{6.2}$$

$$= M_0 - 2.5 *^{10} \log \frac{\mathcal{F}}{\mathrm{T}_{exp}} + \kappa f_z \tag{6.3}$$

[1] If you are interested in more exotic objects, like galaxies or planetary nebulae, you might want to use another shape for your aperture.Just check out the stsdas package.

Here, T_{exp} is the time during which the exposure has been taken. κ is the extinction coëfficient which is dependent of the weather conditions and f_s the airmass. This is a measure for the volume of the column of air in the atmosphere which is between the observer and the star. The airmass is defined to be 1 at the zenith and goes up with the angle (see figure 6.2. Doing photometry in IRAF

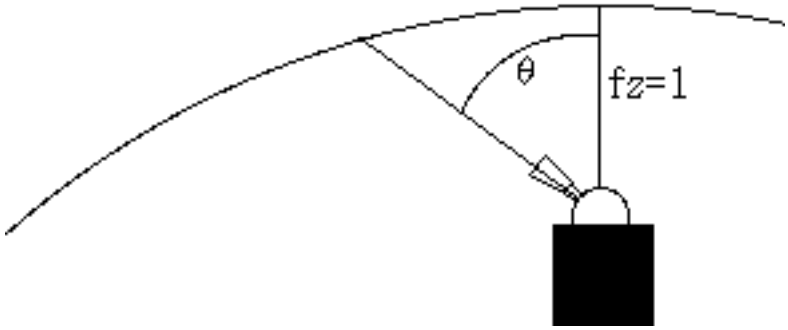

Figure 6.2: The airmass is defined as being 1 in the zenith. The airmass increases with the angle (θ in the image) because there is a larger volume of air between the observer and the object.

is, even in terms of IRAF, quite cumbersome. However, if relative photometry is understood, the extension to absolute photometry is quite easy. The following section will give an introduction to relative photometry. §**??** will introduce photometry when there is a reference star within the field. §6.3 will conclude this chapter by explaining how to do photometry when a separate reference star has been observerd.
For photometry we will be using the `phot` routine, which is part of the `apphot` package. So let's load that package before continuing

```
cl> noao

      artdata.      digiphot.     nobsolete.    onedspec.
      astcat.       focas.        nproto.       rv.
      astrometry.   imred.        observatory   surfphot.
      astutil.      mtlocal.      obsutil.      twodspec.

no> digiphot

      apphot.   daophot.  photcal.  ptools.

di> apphot

      aptest        findpars@     pconvert      polymark      psort
      center        fitpsf        pdump         polypars@     qphot
      centerpars@   fitsky        pexamine      polyphot      radprof
      daofind       fitskypars@   phot          prenumber     wphot
      datapars@     pcalc         photpars@     pselect
ap>
```

6.1 Relative photometry

Relative photometry is used when one is not interested in absolute magnitudes but only in magnitude differences within one science frame. Doing relative photometry is however a very good introduction to doing photometry because it is the most simple way of doing it.
Suppose that you have a reduced CCD image (if not, why did you start reading here anyway? First follow chapter **??**). To be able to do any photometry at all, you will first have to

do some preparations. In this section, we will assume that you are working in the directory `/scratch/username/workdir/` and want to know the relative magnitude of objects in an image called `image.fits`.
So now, let's start with the real work! First start up ds9:

```
ap> !ds9&
```

Now, we first will have to find the positions of the objects. For this we will have to use the imexamine command.

```
ap> imexamine image.fits
```

The image will now be opened in ds9. Remember that if you see just a small part of the image you should tap the `q` key and type (in the xgterm) (or see the first chapter to put it in your login.cl)

```
ap> set stdimage=imt8192
```

, then just restart the `imexamine` procedure (please note that the instructions in section **??** explain you how to let this be a standard setting).
So now you will see your science image on your screen. For photometry you should make a file in which the coördinates of all stars in your file are listed. So just start your favorite text editor and create a file. For this example, the file will be called `starlist.dat`. Now click on the ds9 screen (if a green circle is created, just click it and push the `del` button). You will see that when you move your mouse over the image, it looks like a blinking circle. Now put your mouse cursor on the star and push the `a` button (try to put the cursor as much in the center of the star as you can). IRAF will now fit a 2 dimensional gaussian to your star. The output will look like

```
#   COL     LINE   COORDINATES
#     R     MAG    FLUX      SKY     PEAK     E    PA BETA ENCLOSED   MOFFAT DIRECT
 459.86  628.62 459.86 628.62
  23.98  11.25 317086.   122.3   3109. 0.09   14 5.52     8.43     8.40   7.99
 559.53  594.87 559.53 594.87
  23.71  11.56 237920.   122.2   2398. 0.10   14 2.30     8.63     8.03   7.91
 878.58  646.01 878.58 646.01
  26.61  10.80 480511.   122.5   4696. 0.06   14 5.30     8.42     8.42   8.86
```

. The upper line defines what the numbers mean. For each star two lines of text are used. This example contains two stars. You are only interested of the first two numbers of every star which are the column (COL) and line (LINE) number. In this case the file `starlist.dat` will look like

```
444.747 246.321
357.675 1523.428
770.480 1459.760
```

. Note that you should not use extended sources, but just stars! Now we will have to find out which radii we whould use for the apertures. To do this wil will need to do some examination of our image. To assure that we look at the stars on different frames in the same way, we will need to find a measure which does not depend on the scale of the image or the seeing of the air. The measure we will use for this reason will be the Full Width at Half Maximum (FWHM). When using `imexamine` with the `m` button the FWHM is calculated in three ways. Those values are found under `ENCLOSED`, `MOFFAT` and `DIRECT`. Those are three different routines to calculate the FWHM. Normally those values are close to eachother and you can use the mean value. However, it can happen that you get some values which are at least a bit odd. This is the case in our example. This is probably caused because the source you selected was not a point source and has a size of its own which is added to the seeing which causes a composed FWHM. Because the

FWHM is the same for all point sources along the image, just don't use those weird values in your calculations. You will first have to find out the number of FWHM's you will have to choose for the inner aperture. A very important thing is that you use the same number for all observations sot that you are consistent. Normally it is good to use the FWHM as the unit of your inner aperture radius[2]. In a moment we will show what you can do to find out how to optimize this radius. For the inner radius of the sky you have to choose a value which is not interfering with the star itself. We will also come back to this later on.

So now, let's put all the data we found to use. First tap the `q` button when ds9 is focussed. You will now be able to type in the xgterm again. `phot` is a very extended command. Therfore it will use different parameterfiles to get the right parameters set[3] You recognise those parameter lists because their name (after pressing the **`b`** button) ends with an '@'. We will just continue with our exampling of doing photometry on an image called `image.fits`. First, you will have to specify the image on which photometry should be done.

```
ap> phot.image="image.fits"
```

Then tell phot where it can find the coördinate list.

```
ap> phot.coords="starlist.dat"
```

If you leave the output filename on `default`, the resulting image will be just the name of your image, with the extension `.mag.1` (where the number increases every time you do photometry with the same image).

```
ap> datapars.fwhmpsf=8.3
```

Now we will tell IRAF where in the header of our FITS file it will be able to find some observation specific parameters, or the parameters itself. Those are the readout noise, the gain, the exposuretime, the airmass, date and time of observation and the filter. The last two will not be used in calculations, but it will anyway be quite good to put them here because they will then also be added to the photometry file. For all numbers, you could also put in the value of the number. In table 6.1 the parameters are listed. Empty fields mean that the parameter does not exist. Save

setting	parameter (if in header)	parameter (if number)
CCD readout noise	ccdread	readnoi
CCD gain	gain	epadu
Exposure time	exposur	otime
Airmass	airmass	xairmas
Filter	filter	ifilter
Date and time of observation	obstime	otime

Table 6.1: Photometry parameters for the data.

the parameter list. If you accessed it through **`epar phot`**, you will now return of the parameter list of `phot`. The next parameterlist you should edit is `fitskypars`. You will have to set the inner and outer annulus of the sky aperture. The inner anulus should be a couple of FWHM. For this example, we will just take it to be 5FWHM. More sky means better mean value but has greater chances of including other stars. Less sky gives optimal chances of having only sky, but givces worse counting statistics. It is a trade off between those two and we will give the sky annulus a radius of 3FWHM (which should be an ok value).

[2]When writing NFWHM we mean N times the value for the FWHM, so if the FWHM is 10, then 5FWHM is 50.

[3]When eparing phot, some of the options will have a description which ends by the word "parameters". If you go to them with your cursor, just type **`:e`** to edit that parameter file.

```
ap> fitskypars.annulus=41.5

ap> fitskypars.dannulus=24.9
```

If there are many stars in your image, you should take a look at the section about κσ clipping (6.4). Now you will have to set the photometry parameters. This is accounted for in the `photpars` list. For the moment you only will have to set the size of the aperture which will be put around the star. The number you should choose is practically a couple of FWHM. The idea is that the aperture contains all stellar light and as low sky as possible. Because a star has no well-defined edge, this is impossible to do precisely. The dependence of your result as a function of the aperture size is illustrated in figure 6.3.

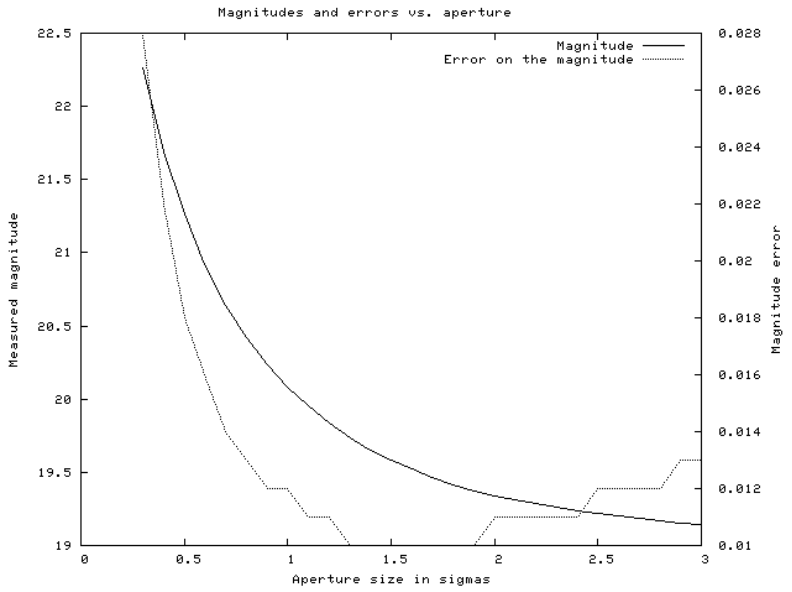

Figure 6.3: Graph of the aperture size vs. the magnitude (solid; left scale) and the error on the magnitude (dotted; right scale). Both are in arbitrary units. The change in the magnitude becomes smaller with bigger apertures. The error has a minimal value. You should choose your parameters close to that minimum, where, as you can see, the magnitude is more or less constant.

As you see, the magnitude change is less for big apertures (after some FWHM you include a bigger percentage of the stellar light; see section **??**). On the other hand, a too big aperture gives relatively big errors because you include more and more sky (noise) in your stellar aperture. Again, this is a tradeoff between an aperture which is too small and one which is too large. To make a good estimate, you can choose to put more numbers in here, separated by comma's. A good rule of thumb is to set a couple of apertures for the first observation and find a number for which the magnitude does not change a lot when you enlarge it and where the error is low. So let's just say we want to do photometry woth an aperture of 1FWHM, 1.5 FWHM ,2 FWHM and 2.5 FWHM.

```
ap> photpars.aperture=4.15,8.3,12.45,16.6
```

Another thing you should do is to set the standard deviation of the sky. Just run `imexamine` again. Put your cursor on some random places where there is clearly no starand press the **m** key.

Do this a couple of times. IRAF will return a list like this one:

```
Log file log.log open
#            SECTION     NPIX     MEAN   MEDIAN   STDDEV      MIN      MAX
[1016:1020,1904:1908]       25    125.7    124.2    12.32    102.2    156.3
[1208:1212,1849:1853]       25    126.3     127.    16.05     92.3    158.6
 [811:815,1765:1769]       25    124.4    123.1    16.83    88.57    155.5
 [827:831,1653:1657]       25    122.5    122.9    12.33    97.63    152.7
[1088:1092,1675:1679]       25    123.8    123.2    14.66    99.42    168.6
 [578:582,1547:1551]       25    122.6     121.    12.94    101.9    150.6
 [346:350,1300:1304]       25    119.5    119.5    9.356    93.54    133.6
```

Take the numbers given in the STDDEV column and calculate their mean value. This is the standard deviation of the sky. In our case, this number is equal to `13.5` so we plug that into our parameter list

`datapars.sigma=13.5`
We are now almost ready to run phot! We just have to be sure that phot will not want to run interactively because this is not supported at all by DS9.

`ap>` **`phot.interac=nophot`**

Now just run phot

`ap>` **`phot`**

You will have to answer a couple of questions first (you basically already answered all questions before so just hit the **`return`** key a couple of times.). Phot will take a moment of contemplationd and, if nothing goes totally wrong, it returns to the command line without any comment. An output file has now been created which is called `image.fits.mag.1` where `image.fits` is the original filename of your observation. If you do photometry for a second time with the same image (because you altered the star list or found out you made some error), a new file will be created which is called `image.fits.2` etc.
Let's look at the contents of the file we just created by editing it in our favourite editor. First note that the first couple of lines (about 55 of them, we will refer to them with the word header) contain all info you could possibly want to know. Actually this is all standard info plus the stuff you plugged into the command. You can always find back which value you gave to which parameter when running the task and you can immediately see if you made an error while plugging it all in. Then you get something of this form

```
#N IMAGE                XINIT     YINIT     ID    COORDS                  LID     \
#U imagename            pixels    pixels    ##    filename                ##      \
#F %-23s                %-10.3f   %-10.3f   %-6d  %-23s                   %-6d
#
#N XCENTER    YCENTER    XSHIFT  YSHIFT  XERR    YERR            CIER CERROR   \
#U pixels     pixels     pixels  pixels  pixels  pixels          ##   cerrors  \
#F %-14.3f    %-11.3f    %-8.3f  %-8.3f  %-8.3f  %-15.3f         %-5d %-9s
#
#N MSKY           STDEV          SSKEW          NSKY   NSREJ     SIER SERROR   \
#U counts         counts         counts         npix   npix      ##   serrors  \
#F %-18.7g        %-15.7g        %-15.7g        %-7d   %-9d      %-5d %-9s
#
#N ITIME          XAIRMASS       IFILTER                OTIME                  \
#U timeunit       number         name                   timeunit               \
#F %-18.7g        %-15.7g        %-23s                  %-23s
```

```
#
#N RAPERT   SUM           AREA       FLUX          MAG    MERR   PIER PERROR   \
#U scale    counts        pixels     counts        mag    mag    ##   perrors  \
#F %-12.2f  %-14.7g       %-11.7g    %-14.7g       %-7.3f %-6.3f %-5d %-9s
#
```

This is the legend. Each variable description consists of three lines. The first line is the description of the variable. The second line explains what is the type of number you would expect and the third line describes the variable in c-code (i.e. `%-11.7f` means that your number is a float (`f`) which consists of maximal 11 numbers before and 7 numbers behind the decimal point). For photometry, you are actually just interested in the last lines of data and the coordinates, for bookkeeping purposes.

```
image.fits             444.870   246.210   1     starlist.dat          1      \
  (...)
   4.15     24151.        54.33944   17543.44      20.385 0.010 0    NoError  *\
   8.30     61064.88      216.6889   34715.98      19.644 0.008 0    NoError  *\
   12.45    100020.6      487.1443   40784.94      19.469 0.009 0    NoError  *\
   16.60    148746.6      865.7964   43467.62      19.399 0.011 0    NoError  *
image.fits             357.450   1523.240  2     starlist.dat          2      \
   (...)
   4.15     27136.17      54.51937   20534.05      20.214 0.009 0    NoError  *\
   8.30     67444.49      216.8462   41185.11      19.458 0.007 0    NoError  *\
   12.45    107615.9      487.2191   48615.23      19.278 0.008 0    NoError  *\
   16.60    156338.6      865.8539   51486.48      19.216 0.010 0    NoError  *
image.fits             770.480   1459.760  3     starlist.dat          3      \
   (...)
   4.15     14350.09      54.14149   7794.688      21.265 0.018 0    NoError  *\
   8.30     49352.14      216.4543   23144.06      20.084 0.012 0    NoError  *\
   12.45    95783.91      487.1623   36798.76      19.580 0.010 0    NoError  *\
   16.60    150796.8      866.3364   45901.65      19.340 0.011 0    NoError  *
```

The image is called image.fits, the numbers next to it are the coördinates of the stars and their ID (just numbered 1, 2 and 3). Then the name of the coördinate list is shown and another labelling ID (which for some obscure reason is the same as the other one). For the last lines, the first number is the radius of the aperture (you see 4 lines because we specified 4 aperture sizes before), the second is the number of counts in the aperture, the third is the area of the aperture. Then the calculated flux and the magnitude calculated based on that knowlege and its error. The last two variables have to be `0` and `NoError`. If they are not, an error occured when doing photometry and you really should get help (or do it all over again). Note that phot takes the integration time to be 1. For the moment this is not important. It might be good to rewrite equations 6.1 and 6.3 to

$$\mathcal{F} = \mathrm{SUM} - \mathrm{MSKY} \times \mathrm{AREA} \tag{6.4}$$

$$M_{\mathrm{IRAF}} = M_0 - 2.5 *^{10} \log \frac{\mathcal{F}}{1.(\mathrm{T}_{exp} \equiv 1)} \tag{6.5}$$

Just note that the extinction due to the earth atmosphere is put to 0. The zeropoint is put to an arbitrary number given by the ZMAG keyword in the header. Now, let's just calculate the difference between two objects in our frame, which is actually the goal of this section. Let us first see hoe relative magnitudes in the science fram relate to real magnitude differences by deriving this formula using equation 6.3

$$\begin{aligned}
\Delta M \equiv M_1 - M_2 &= M_0 - 2.5 *^{10} \log \mathcal{F}_1 + 2.5 *^{10} \log \mathrm{T}_{exp} + \kappa f_z && (6.6)\\
&- (M_0 - 2.5 *^{10} \log \mathcal{F}_2 + 2.5 *^{10} \log \mathrm{T}_{exp} + \kappa f_z) && (6.7)\\
&= (M_0 - 2.5 *^{10} \log \mathcal{F}_1) - (M_0 - 2.5 *^{10} \log \mathcal{F}_2) && (6.8)\\
&= M_{\mathrm{IRAF1}} - M_{\mathrm{IRAF2}} && (6.9)
\end{aligned}$$

Because both images are taken on the same moment of the night, and have the same airmass (because the frame is relatively small), there is no difference in airmass and extinction coëfficient. Of course, the exposure time of both objects is the same. So a preliminary conclusion is that the relative magnitude difference is equal to the difference in the numbers calculated by IRAF. The problem is that if we want to do correct error analysis, we will still have to take the error on the zero point into account. Therefore, you always have to gauge for the absolute magnitude to find good values.

6.2 Field photometry

Field photometry is a very good tool to use when your science frame contains a lot of stars. For this method, you will have to look up the stars in your science frame. The problem with this method is that you will have to use a standard photometric system (e.g. Johnson) because you need a good all sky catalog. For example, if you use the Johnsonm photometric system the US military database of stars[4] is one of those databases. This is however not really precise, so you need at least 10 stars to do a good photometric callibration. Some of the filters of the Mercator are from the geneva system which is not very standard, so there are no good catalogs on the Internet. A good catalog for the Geneva system by Rufener is available at the library, the book is from 1987 [5] but is already quite complete (however an update came out in 1999 which might be a good idea for the institute to buy[6]). There are methods to convert convert from geneva to Johnson and back. This is quite a job and has nothing to do with IRAF so we will just refer to one of them (**?**). To find which star on your frame is which star in the catalog, you might just want to callibrate your science frame with a world coördinate system. See §5.
So now you have some reference stars from a catalog. First let us do some photometry with them following the procedure described above. So now you have IRAF magnitudes for you standard stars and for the ones you want to do photometry with. The absolute magnitude you want to find can now been calculated in a trivial way

$$M_{catalog} = M_{ref} + C \tag{6.10}$$
$$M_{sci} = M_{\mathrm{IRAF}} + D \tag{6.11}$$
$$\tag{6.12}$$

$M_{catalog}$ is the magnitude found in the catalog, M_{ref} is the IRAF magnitude of the standard star, C and D are some constants, M_{sci} is the absolute magnitude which you are looking for, M_{IRAF} is the magnitude found by IRAF. According to equation 6.9,

$$M_{sci} - M_{catalog} = M_{\mathrm{IRAF}} - M_{ref} \tag{6.13}$$
$$\Longrightarrow \quad C = D \tag{6.14}$$

So now, you can find C (and so D) by just subtracting the measured value and the catalog value of the magnitude from the measured value for your reference star. Do this for each star. Now you will have a couple of values for C. Take the mean value foe C and adjust the zero magnitude of the phot package according to it. As an example if you run phot for the first time the standard value of the zero point is 25. Let's say that you found a C of 5. Now you will have to change your zero point magnitude to 20 and do the photometry again.

```
ap> photpars.zmag=20
ap> phot
```

You will have to do this over and over until the value you find for C is more or less 0. The

[4] http://www.nofs.navy.mil/data/fchpix/
[5] library string: XXII-A-Ruf-1
[6] yes this is a hint

error on the magnitude found by IRAF (for the star on which you are doing science) is given by the software itself. The error on the zero point is given by the highest deviation of the mean value (the maximal error). The three errors add up quadratically ($Err_{tot} = \sqrt{Err^2_{zero} + Err^2_{Machine}}$)

6.3 Reference star photometry

Reference star photometry is done by taking a science exposure and an exposure of an object of which you want to do photometry. Single object photometry differs a lot from crowded field photometry and relative photometry because you can not just let all constants drop out. So we will have to do some mathematics.
All calculations are done with the help of the header of a FITS file which contains all data IRAF can plug in the formulas.So let's just have a look at the header of our FITS file in ds9

```
ap> !ds9&
```

Now grab your mouse and click "File", in that menu click "Open". Browse to your image file, click on it and press "ok" (note that you can not use the `display` command in IRAF because of some unclear reasons). When the image is loaded, click the "HEADER" button. You will now see the header. A header line consists of a KEY (in capitals, displayed in blue) a "=" sign which is followed by a value. Behind the value there can be a "/" followed by a description of the header keyword. Find out what are the header keywords for airmass (in the middle of the exposure, for statistical reasons) and exposure time are. Now go back to IRAF and edit two parameters of the phot command

```
ap> phot.exposur="TEXP"
ap> phot.airmass="FZ_MP"
```

Here I assume that the header keyword for the exposuretime is "TEXP" and the one for the airmass is "FZ_MP" (which are the actual values at MERCATOR). Furthermore you will have to find the zero point of the magnitude scale like you did before. You assume that the zero point does not change too dramatically in time, so the zero point of your callibrator is the same as that of your science frame. Because the magnitude of your callibrator is rather well known, it is ok to take the zero point based on this one star, or multiple observations of it. The error on the zero point of the magnitude scale is then given by the incertainty on the magnitude of your callibrator. For the sake of the example I will take this value to be 20.5

```
ap> phot.zmag=20.5
```

Now just follow the whole phot procedure again.
Let us recall the formulas for the absolute photometry and for the magnitude found by IRAF

$$M_{\mathrm{IRAF}} = M_0 - 2.5 *^{10} \log \frac{\mathcal{F}}{\mathrm{T}_{exp}} \tag{6.15}$$

$$M_{real} = M_0 - 2.5 *^{10} \log \frac{\mathcal{F}}{\mathrm{T}_{exp}} + \kappa f_z \tag{6.16}$$

$$= M_{\mathrm{IRAF}} + \kappa f_z \tag{6.17}$$

Where M_{real} is the real magnitude of the object. There are two constants in this formula which are not dependent of the science frame. Of those constants, we already plugged in one (the zero point of the magnitude scale). The extinction coëfficient κ can not really been measured just one time per month because it varies over a time scale of nights. But because it is the only unknown value in the formula we can rather easily find it

$$\kappa = \frac{M_{real} - M_{\mathrm{IRAF}}}{f_z} r \tag{6.18}$$

κ	probability that it is a signal
1	68.3 %
2	95.4 %
3	99.73 %
4	99.99 %
5	99.9999998 %
6	99.9999999997 %

For this it comes out handy that you put the header entry for the airmass in phot's parameter list because it will put the air mass in the magnitude file, under parameter "XAIRMASS". So now we have all data to find κ. If we plug in this parameter in equation 6.17 we now have an equation which yields the real magnitude when the IRAF magnitude is known.

6.4 κσ clipping

It could happen that your sky aperture (partially) contains another star (see image 6.4). To prevent the sky to give a far too high value, you would like to correct for this. The method used for this is called κσ clipping. The κ is just some positive number (note that this κ has nothing to do

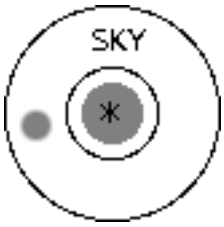

Figure 6.4: Schematic figure of a photometry aperture with a star within the sky radius.

with the extinction coëfficient) and σ is the standard deviation of the noise. If one sees the noise as a gassian distribution, the number of σs a signal is away from the mean value reduces the odds of it just being a large noise peak. In table 6.4 the probability that a peak is due to data rather than from noise is shown. Note that when you take more σs, you might also drop data points which just happen to be very low. One could argue that the ideal number for sigma is very huge. Although it is true that the odds of a noise peak being smaller than 1000σ are very, very small, so everything of this size is almost certainly data. The problem with this argument is that a lot of data will also be rejected because its value is not large enough. 3σ is ok for most of the applications, or to claim a "discovery". If, however, you are looking for big amounts of stars ($\gg 100$) you might want to crank up the precision to more than 3σ. The default value of κ is 3. If you want to change this value, you will have to change two parameters which define the upper (how many σs more than average) and lower limit (how many σs less than average) of your data. Normally you just can take them the same, but we will show you how to change them both so that you know where to find them. If, for example, you would like to reject all data which is 5σ larger or 4σ smaller than the mean value.

```
cl> noao

no> digiphot

di> apphot

ap> fitskypars.sloreject=4.
```

```
ap> fitskypars.shireject=5.
```

When you run the standard photometry procedures, any "object" which is less than 4σ larger than the mean of the noise (sky), will be rejected as well as every objevt which is less than 5σ than the sky.

Appendix A

UNIX

Although this manual describes the use of IRAF this chapter will describe some useful UNIX commands. When you work on a lot of observational data it saves a lot of time to work with lists of filenames in stead of filenames themselves. In this chapter we will introduce some basic UNIX commands to you. We strongly recommend checking the UNIX command-line out, it will save you a lot of time in the long run. This chapter is not meant to describe everything the UNIX command-line has to offer — whole books have been written describing it or its commands.

A.1 Getting help on the command-line using `man`

Although the use of the `man` command is very simple it is so important that we have given it its own section. You can get help for most if not all UNIX command line programs / commands using `man`. Typing the command

```
> man somecommand
```

will drop you into a help reader for the `somecommand` manual pages (hence `man`, see figure A.1). If there is no help available for `somecommand` then `man` command will let you know:

```
No manual entry for somecommand
```

You can quit the help reader by pressing **`q`** or by reaching the end of the help text. When you quit the reader you will be dropped back at the command line. Scrolling through the manual text is done using the **`space bar`** or the **`Page Up`** or **`Page Down`** keys.

A.2 Chaining commands and reading and text files

When you want to read through a text file you can use the `cat` command the usage is as follows

```
> cat blah.txt
```

this will show the contents of the file `blah.txt` in one go, so you terminal might scroll if the text file is long. If you want to read through the file on a page by page basis you should use the `more` command. Just type

```
> more blah.txt
```

to read through the file `blah.txt`. Pressing space will scroll to the next page when you are

Figure A.1: A terminal running `man`, showing the manual entry for the `chmod` command.

using `more`. The `cat` command is mostly useful when combined with other commands. For instance, if you want to look for the word *Monday* in that text file you can use the command;

```
> cat blah.txt | grep "Monday"
```

This will output any line that contains the word *Monday*. The `|` character in the command sends the output of one program to another program — in this case from `cat` to `grep`. In this context `|` is called a pipe. If you want to write the output from some command to a file you can use the `>` character. Lets say you want only want all the lines containing *Monday* to be written to a new text file type the following;

```
> cat blah.txt | grep "Monday" > output.txt
```

This command will go through the file `blah.txt` find any line that contains *Monday* write those lines to the text file `output.txt`. Note that this overwrites the file `output.txt`. You can change the `>` character into a `>>` character, then the lines will be appended to the file `output.txt` if it already exists. If you want to look through several files you can use the `fgrep` command. To go through all files with a `.txt` extension (the part behind the last `.` in the filename) and search for the word *Monday* you can type the following;

```
> fgrep "Monday" *.txt
```

This will again only output the lines that contain the word *Monday* except this time the output will start with the name of the file where that the line was found in.

When you want to quickly read through the first or last few lines of a text file you can use the `head` and `tail` commands. The way you use them is

```
> head file.txt
```

or

```
> tail file.txt
```

to read through the respectively the first few or last few lines of the file `file.txt`. Using

```
> head -5 file.txt
```

will show only the first 5 lines of that file (`tail` works the same way).

A.3 See what processes are running and how to stop them

On a UNIX system it is possible to find out what processes (programs etc.) are running on your computer. To get a list of the processes running type the command;

```
> ps
```

the list you get shows a column called `PID`, in this list are the process IDs. In the last column, the `CMD` column, there are the commands to which the processes belong. Don't be confused since the column may show processes that were started automatically. If a program is hung sometimes the only way it can be stopped is by using the `kill` command. The kill command takes a `PID` as parameter, the command

```
> kill 1000
```

will try to kill the process with `PID` 1000. Sometimes this will not work and you have to use more force;

```
> kill -9 1000
```

Note that you might not be able to kill every process because some might be run by other users, UNIX will generally not allow you to touch what other users are doing (unless you are `root`). The `ps` command takes many parameters. For instance, if you want to see all the processes being run by some user use the `-u` parameter. The processes run by user `tcoenen` can be listed using;

```
> ps -u tcoenen
```

The other way of getting a quick and updated in real time view of what is running on your computer is by using the `top` command. With the command

```
> top
```

The list that `top` shows will update every few seconds and also show the CPU use, which is nice (see figure A.2). You can then quit `top` by pressing the **`q`** key.

A.4 A few examples of using `awk`

The UNIX commands `awk`, `paste` and `sort` are helpful in dealing with files that contain columns. Since IRAF commands usually take as inputs lists of files the aforementioned UNIX commands are very useful in combination with IRAF. Let's start with the most versatile and hardest command; `awk`. The `awk` command is not actually a command it is a whole language, so we will not treat it in full.

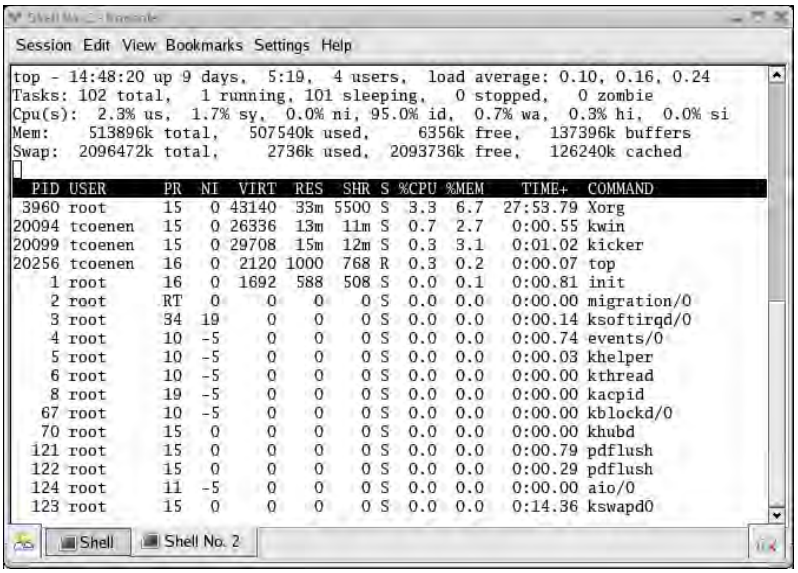

Figure A.2: A terminal running `top`.

A.4.1 Copying lots of files while changing their names

Let's say that you have a directory with images named `obs001.fits`, `obs002.fits` etc. and you want to copy them to files called `obs001b.fits`, `obs002b.fits` etc. You can do this by hand but that will result in lots of typing (something you don't want). First we will have to tell `awk` what all the filenames are that can be done with `ls` (see below) or by creating a text file that contains a list of filenames (one on each line, nothing fancy) and using cat and a pipe to send that list to `awk`. Now we are going to use `awk` to create a batch file with lines like;

```
cp obs001.fits obs001b.fits
```

You do this with the command

```
> ls *.fits | awk '{split($0,array,".fits"); print "cp",$0,array[1]"b.fits"}' > batch
```

OK, so this is not a very pretty command but it creates exactly the lines we were looking for and writes them to the file `batch`. The part behind `awk` enclosed between `''` is the actual command that `awk` runs for each line it receives (in this case from `ls`). Here it runs first the split function on each line, the `$0` means that split will work on the whole line `awk` receives[1]. The second parameter of for the `split` function is the name of the array that is created to hold the output of the `split` function. The last parameter for the split function is the text where the split is made, in this case `.fits`. The trailing `;` is there to separate the function calls (think C, C++, ECMAscript or whatever). The second part of the command creates the actual text lines, `print` tells `awk` to print what follows after it. Pieces of text between `""` will just be printed as they appear so `print "blah"` will result in `blah` appearing. The comma `,` creates a space in the printed output. Again `$0` is the whole line `awk` received, here just one filename per line. Now comes the hard part `array[1]` takes the array created by `split` (again on a line by line basis) and takes the first element of that array. Since `split` was called with `".fits"` as its last argument the first element of `array` will be something like `obs001` (if the filename was `obs001.fits`). Note that when using `awk` the first element of an array is not the element with index 0 but the element with index 1.

[1] It can also work with only one column, which will be described later.

Although it may look ugly `array[1]"b.fits"` results in something like `obs001b.fits`. There is no comma separating `array[1]` and `"b.fits"` because that would result in a space between them.

So now we have a text file with lines like

```
cp obs001.fits obs001b.fits
```

but we will still have to run this file to do the actual copying. First we have to make the batch file executable (in UNIX text files are not executable by default). Use `chmodchmod@chmod` in the following fashion to make the file `batch` executable;

```
> chmod u+x batch
```

Now that `batch` is executable we run it with the command:

```
> ./batch
```

Each line in the batch file will be executed as if it was typed from the command line. You just saved yourself a lot of typing, which is a good thing.

A.4.2 `awk` and tables

In a footnote we promised to explain the use of `$0` in `awk`. By itself `awk` already has a concept of columns. When `awk` receives a line that line is available as `$0`. But if the line contains columns then each column is also available as `$1` for the first column `$2` for the second column etc. This can be used to for instance create LaTeXtables easily. That will be my third `awk` example, but we will first show you how to select just a few columns from a text file.

Assume we are dealing with a text file that contains the following table (text files with columns separated by tabs);

```
x1 x2 x3 x4
10 12 13 41
22 13 53 73
23 91 88 39
```

Now let's say that we only want the `x1` and `x3` columns, that is possible with the command

```
> cat tabelin.txt | awk '{print $1"\t"$3}' > tabeluit.txt
```

. This command writes the columns `x1` and `x3` to the file `tabeluit.txt`. The `\t` creates a tab character in the output. With just a small extension this can be used to create the meat of a LaTeX table (you will still need to provide the opening and closing for that table). The command

```
> cat tabelin.txt | awk '{$1,"&",$2,"&",$3,"&",$4,"\\\\"}' > table.tex
```

will create the following text in the file `table.tex`;

```
x1 & x2 & x3 & x4 \\
10 & 12 & 13 & 41 \\
22 & 13 & 53 & 73 \\
23 & 91 & 88 & 39 \\
```

The `awk` command contains 4 backslashes even though `LaTeX` only needs 2 backslashes. This is because within `awk` 1 backslash has a special meaning. By typing two backslashes in succession `awk` understands that you just mean a normal backslash[2].

[2]This is an instance of escaping the backslash character, every programming language has a concept like it —

A.5 Sorting a table using `sort`

Some of you may have sorted through tab separated files using, for instance, Gnumeric (a spreadsheet). Although that is possible it is nice to know that UNIX contains the `sort` command that will also let you sort through text files from the command line. Sort can sort the file by any column using the command;

```
> sort -k1 somefile.txt
```

This command will sort using the first column (as specified by the `1` directly after `-k`) the file `somefile.txt` and write the output to the terminal. The above command actually does string compares, but when you are dealing with numbers in their ASCII representations it is safer to tell `sort` explicitly that the column contains numbers. The `-n` option lets you do that;

```
> sort -k1 -n somefile.txt
```

Again this command sorts the file `somefile.txt` and writes the sorted file to your terminal but this time the first column is treated as containing numbers. If you only want unique entries — that is if you want the sorted file to never contain the same sort index twice or more — you can use the `-u` option. So

```
> sort -k1 -n -u somefile.txt
```

sorts the file `somefile.txt` using the first column, treated as containing numbers, creates output that only contains unique lines and prints it to the terminal. By appending`> sorted.txt` to the previous commands you can create sorted files (the use of the `>` character is explained in section A.2). The `-r` option of sort performs the sort in reverse.

A.6 Pasting columns together using `paste`

When you have several tab separated text files that contain columns that you would like to join there are two ways to do it. If you have two files that have the same columns then you can create one large file containing all the rows from the two files. This is easy you can do that using the following two commands;

```
> cp file1.txt all.txt
```

```
> cat file2.txt >> all.txt
```

The first command here (the copy or `cp` command) creates a straight copy of `file1.txt` to `all.txt` and the second command appends the contents of `file2.txt` to that copy. Alternatively you can also do it in one go;

```
> cat file1.txt file2.txt > all.txt
```

But what if you have a two files with different columns that you want to combine in such a way that the combined file contains all the columns from these two files? Then just appending the first file won't cut it. Using the `paste` command it is possible to make a file that contains all the different columns from the two files that you want to combine. Using paste is easy the command

not necessarily for the backslash though. HTML, for instance, needs to escape the `<` and `>` characters because all HTML tags are enclosed within these characters. So HTML needs to decide whether you meant to use a tag or whether you meant to just use text containing `<` and `>`.

```
> paste file1.txt file2.txt
```

will output to the terminal all the columns of `file1.txt` and `file2.txt` next to each other. By appending `> allcolumns.txt` to the previous command you can create a files to hold the output of `paste`.

Index