GNU Astronomy Utilities

Astronomical data manipulation and analysis programs and libraries
for version 0.2, 3 October 2016

Mohammad Akhlaghi

Gnuastro (source code and book) authors (sorted by number of commits in project history):

Mohammad Akhlaghi (akhlaghi@gnu.org, 455)
Mosè Giordano (mose@gnu.org, 29)
Vladimir Markelov (vmatroskin@gmail.com, 12)

This book documents version 0.2 of the GNU Astronomy Utilities (Gnuastro). Gnuastro provides various programs and libraries for astronomical data manipulation and analysis.

For myself, I am interested in science and in philosophy only because I want to learn something about the riddle of the world in which we live, and the riddle of man's knowledge of that world. And I believe that only a revival of interest in these riddles can save the sciences and philosophy from narrow specialization and from an obscurantist faith in the expert's special skill, and in his personal knowledge and authority; a faith that so well fits our 'post-rationalist' and 'post-critical' age, proudly dedicated to the destruction of the tradition of rational philosophy, and of rational thought itself.

—*Karl Popper. The logic of scientific discovery. 1959.*

Short Contents

Table of Contents

1 Introduction

GNU Astronomy Utilities (Gnuastro) is an official GNU package consisting of separate programs and libraries for the manipulation and analysis of astronomical data. All the programs share the same basic command-line user interface for the comfort of both the users and developers. Gnuastro is written to comply fully with the GNU coding standards so it integrates finely with the GNU/Linux operating system. This also enables astronomers to expect a fully familiar experience in the source code, building, installing and command-line user interaction that they have seen in all the other GNU software that they use. The official and always up to date version of this book (or manual) is freely available under Appendix C [GNU Free Documentation License], page 261, in various formats (pdf, html, plain text, info, and as its Texinfo source) at `http://www.gnu.org/software/gnuastro/manual/`.

For users who are new to the GNU/Linux environment, unless otherwise specified most of the topics in Chapter 3 [Installation], page 24, and Chapter 4 [Common program behavior], page 45, are common to all GNU software, for example installation, managing command-line options or getting help (also see Section 1.6 [New to GNU/Linux?], page 7). So if you are new to this empowering environment, we encourage you to go through these chapters carefully. They can be a starting point from which you can continue to learn more from each program's own manual and fully benefit from and enjoy this wonderful environment. Gnuastro also comes with a large set of libraries, so you can write your own programs using Gnuastro's building blocks, see Section 10.1 [Review of library fundamentals], page 189, for an introduction.

Finally it must be mentioned that in Gnuastro, no change to any program will be released before it has been fully documented in this here first. As discussed in Section 1.2 [Science and its tools], page 2, this is the founding basis of the Gnuastro.

1.1 Quick start

Gnuastro has three mandatory dependencies and two optional ones for added functionality, see Section 3.1 [Dependencies], page 24. The latest official release tarball is always available as `gnuastro-latest.tar.gz` (`http://ftp.gnu.org/gnu/gnuastro/gnuastro-latest.tar.gz`). For better compression (faster download), we also provide an Lzip compressed tarball at `gnuastro-latest.tar.lz` (`http://ftp.gnu.org/gnu/gnuastro/gnuastro-latest.tar.lz`), see Section 3.2.1 [Release tarball], page 30, for more details on the tarball release. If you have downloaded the tarball in the `TOPGNUASTRO` directory and the dependencies are installed, you can unpack, compile, check and install Gnuastro with the following commands. If you use GNU Tar, the same command (`$ tar xf`) can also be used to unpack `.tar.lz` tarballs.

```
$ cd TOPGNUASTRO
$ tar xf gnuastro-latest.tar.gz    # This works on '.tar.lz' too.
$ cd gnuastro-X.X                  # Replace X.X with version number.
$ ./configure
$ make
$ make check
$ sudo make install
```

See Section 3.3.4 [Known issues], page 43, if you confront any complications. For each program there is an 'Invoke ProgramName' sub-section in this book which explains how the programs should be run on the command-line. You can read it on the command-line by running the command `$ info astprogname`, see Section 1.4 [Naming convention], page 5, and Section 4.6 [Getting help], page 57. The 'Invoke ProgramName' sub-section starts with a few examples of each program and goes on to explain the invocation details. In Chapter 2 [Tutorials], page 14, some real life examples of how these programs might be used are given.

1.2 Science and its tools

History of science indicates that there are always inevitably unseen faults, hidden assumptions, simplifications and approximations in all our theoretical models, data acquisition and analysis techniques. It is precisely these that will ultimately allow future generations to advance the existing experimental and theoretical knowledge through their new solutions and corrections.

In the past, scientists would gather data and process them individually to achieve an analysis thus having a much more intricate knowledge of the data and analysis. The theoretical models also required little (if any) simulations to compare with the data. Today both methods are becoming increasingly more dependent on pre-written software. Scientists are dissociating themselves from the intricacies of reducing raw observational data in experimentation or from bringing the theoretical models to life in simulations. These 'intricacies' are precisely those unseen faults, hidden assumptions, simplifications and approximations that define scientific progress.

> Unfortunately, most persons who have recourse to a computer for statistical analysis of data are not much interested either in computer programming or in statistical method, being primarily concerned with their own proper business. Hence the common use of library programs and various statistical packages. ... It's time that was changed.
>
> —*F. J. Anscombe. The American Statistician, Vol. 27, No. 1. 1973*

Anscombe's quartet[1] demonstrates how four data sets with widely different shapes (when plotted) give nearly identical output from standard regression techniques. Anscombe argues that "Good statistical analysis is not a purely routine matter, and generally calls for more than one pass through the computer". Anscombe's quartet can be generalized to say that users of a software cannot claim to understand how it works only based on the experience they have gained by frequently using it. This kind of subjective experience is prone to very serious mis-understandings about what it really does behind the scenes and can be misleading. This attitude is further encouraged through non-free software[2]. This approach to scientific software only helps in producing dogmas and an "obscurantist faith in the expert's special skill, and in his personal knowledge and authority"[3].

> Program or be programmed. Choose the former, and you gain access to the control panel of civilization. Choose the latter, and it could be the last real choice you get to make.

[1] `http://en.wikipedia.org/wiki/Anscombe%27s_quartet`

[2] `https://www.gnu.org/philosophy/free-sw.html`

[3] Karl Popper. The logic of scientific discovery. 1959. Larger quote is given at the start of the PDF (for print) version of this book.

—Douglas Rushkoff. Program or be programmed, O/R Books (2010).

It is obviously impractical for any one human being to gain the intricate knowledge explained above for every step of an analysis. On the other hand, scientific data can be very large and numerous, for example images produced by telescopes in astronomy. This requires very efficient algorithms. To make things worse, natural scientists have generally not been trained in the advanced software techniques, paradigms and architecture that are taught in computer science or engineering courses and thus used in most software. The GNU Astronomy Utilities are an effort to tackle this issue.

Gnuastro is not just a software, this book is as important to the idea behind Gnuastro as the source code (software). This book has tried to learn from the success of the "Numerical Recipes" book in educating those who are not software engineers and computer scientists but still heavy users of computational algorithms, like astronomers. There are two major differences: the code and the explanations are segregated: the code is moved within the actual Gnuastro software source code and the underlying explanations are given here. In the source code every non-trivial step is heavily commented and correlated with this book, it follows the same logic of this book, and all the programs follow a similar internal data, function and file structure, see Section 11.6 [Program source], page 244. Complementing the code, this book focuses on thoroughly explaining the concepts behind those codes (history, mathematics, science, software and usage advise when necessary) along with detailed instructions on how to run the programs. At the expense of frustrating "professionals" or "experts", this book and the comments in the code also intentionally avoid jargon and abbreviations. The source code and this book are thus intimately linked, and when considered as a single entity can be thought of as a real (an actual software accompanying the algorithms) "Numerical Recipes" for astronomy.

The other major and arguably more important difference is that "Numerical Recipes" does not allow you to distribute any code that you have learned from it and the book is not freely available. So while it empowers the privileged individual who has access to it, it exacerbates social ignorance. For example it does not allow you to release your software's source code if you have used their codes, you can only publicly release binaries (a black box) to the community. Exactly at the opposite end of the spectrum, Gnuastro's source code is released under the GNU general public license (GPL) and this book is released under the GNU free documentation license. You are therefore free to distribute any software you create using parts of Gnuastro's source code or text, or figures from this book, see Section 1.3 [Your rights], page 4. While developing the source code and this book together, the developers of Gnuastro aim to impose the minimum requirements on you (in computer science, engineering and even the mathematics behind the tools) to understand and modify any step of Gnuastro if you feel the need to do so, see Section 11.1 [Why C programming language?], page 236, and Section 11.2 [Program design philosophy], page 238.

Imagine if Galileo did not have the technical knowledge to build a telescope. Astronomical objects could not be seen with the Dutch military design of the telescope. In the beginning of his "The Sidereal Messenger" (1610) he cautions the readers on this issue and instructs them on how to build a suitable instrument: without a detailed description of "how" he made his observations, no one would believe him. The same is true today, science cannot progress with a black box. Before he actually saw the moons of Jupiter, the moun-

tains on the Moon or the crescent of Venus, he was an anti-Copernican and was "evasive" to Kepler[4]. Science is not independent of its tools.

Bjarne Stroustrup (creator of the C++ language) says: "Without understanding software, you are reduced to believing in magic". Ken Thomson (the designer or the Unix operating system) says "I abhor a system designed for the 'user' if that word is a coded pejorative meaning 'stupid and unsophisticated'." Certainly no scientist (user of a scientific software) would want to be considered a believer in magic, or 'stupid and unsophisticated'. However, this can happen when scientists get too distant from the raw data and are mainly indulging themselves in their own high-level (abstract) models (creations). For example, roughly 5 years before special relativity and about two decades before quantum mechanics fundamentally changed Physics, Kelvin is quoted as saying:

> There is nothing new to be discovered in physics now. All that remains is more and more precise measurement.
>
> —*William Thomson (Lord Kelvin), 1900*

A few years earlier, in a speech Albert. A. Michelson said:

> The more important fundamental laws and facts of physical science have all been discovered, and these are now so firmly established that the possibility of their ever being supplanted in consequence of new discoveries is exceedingly remote.... Our future discoveries must be looked for in the sixth place of decimals.
>
> —*Albert. A. Michelson, dedication of Ryerson Physics Lab, U. Chicago 1894*

If scientists are considered to be more than mere "puzzle solvers"[5] (simply adding to the decimals of known values or observing a feature in 10, 100, or 100000 more galaxies or stars, as Kelvin and Michelson clearly believed), they cannot just passively sit back and uncritically repeat the previous (observational or theoretical) methods/tools on new data. Today there is a wealth of raw telescope images ready (mostly for free) at the finger tips of anyone who is interested with a fast enough internet connection to download them. The only thing lacking is new ways to analyze this data and dig out the treasure that is lying hidden in them to existing methods and techniques.

> New data that we insist on analyzing in terms of old ideas (that is, old models which are not questioned) cannot lead us out of the old ideas. However many data we record and analyze, we may just keep repeating the same old errors, missing the same crucially important things that the experiment was competent to find.
>
> —*E. T. Jaynes, Probability theory, the logic of science. 2003.*

1.3 Your rights

The paragraphs below, in this section, belong to the GNU Texinfo[6] manual and are not written by us! The name "Texinfo" is just changed to "GNU Astronomy Utilities" or "Gnuastro" because they are released under the same licenses and it is beautifully written to inform you of your rights.

---

[4] Galileo G. (Translated by Maurice A. Finocchiaro). *The essential Galileo*. Hackett publishing company, first edition, 2008.

[5] Thomas S. Kuhn. *The Structure of Scientific Revolutions*, University of Chicago Press, 1962.

[6] Texinfo is the GNU documentation system. It is used to create this book in all the various formats.

GNU Astronomy Utilities is "free software"; this means that everyone is free to use it and free to redistribute it on certain conditions. Gnuastro is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of Gnuastro that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of the programs that relate to Gnuastro, that you receive the source code or else can get it if you want it, that you can change these programs or use pieces of them in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of the Gnuastro related programs, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for the programs that relate to Gnuastro. If these programs are modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

The precise conditions of the licenses for the programs currently being distributed that relate to Gnuastro are found in the GNU General Public license (`http://www.gnu.org/copyleft/gpl.html`) that accompany them. This book is covered by the GNU Free Documentation License (`http://www.gnu.org/copyleft/fdl.html`).

1.4 Naming convention

Gnuastro is a package of independent programs and a collection of libraries, here we are mainly concerned with the programs. Each program has an official name which consists of one or two words, describing what they do. The latter are printed with no space, for example NoiseChisel or ImageCrop. On the command-line, you can run them with their executable names which start with an `ast` and might be an abbreviation of the official name, for example `astnoisechisel` or `astimgcrop`, see Section 3.3.1.3 [Executable names], page 40.

We will use "ProgramName" for a generic official program name and `astprogname` for a generic executable name. In this book, the programs are classified based on what they do and thoroughly explained. An alphabetical list of the programs that are installed on your system with this installation are given in Appendix A [Gnuastro programs list], page 256. That list also contains the executable names and version numbers along with a one line description.

1.5 Version numbering

Gnuastro can have two formats of version numbers, for official and unofficial releases. Official Gnuastro releases are announced on the `info-gnuastro` mailing list, they have a version control tag in Gnuastro's development history, and their version numbers are formatted like "`A.B`". `A` is a major version number, marking a significant planned achievement (for example see Section 1.5.1 [GNU Astronomy Utilities 1.0], page 6), while `B` is a minor version

number, see below for more on the distinction. Note that the numbers are not decimals, so version 2.34 is much more recent than version 2.5, which is not equal to 2.50.

Gnuastro also allows a unique version number for unofficial releases. Unofficial releases can mark any point in Gnuastro's development history. This is done to allow astronomers to easily use any point in the version controlled history for their data-analysis and research publication. See Section 3.2.2 [Version controlled source], page 31, for a complete introduction. This section is not just for developers and is very straightforward, so please have a look if you are interested in the cutting-edge. This unofficial version number is a meaningful and easy to read string of characters, unique to that particular point of history. With this feature, users can easily stay up to date with the most recent bug fixes and additions that are committed between official releases.

The unofficial version number is formatted like: `A.B.C-D`. `A` and `B` are the most recent official version number. `C` is the number of commits that have been made after version `A.B`. `D` is the first 4 or 5 characters of the commit hash number[7]. Therefore, the unofficial version number '`3.92.8-29c8`', corresponds to the 8th commit after the official version `3.92` and its commit hash begins with `29c8`. The unofficial version number is sort-able (unlike the raw hash) and as shown above is very descriptive of the state of the unofficial release. Of course an official release is preferred for publication (since its tarballs are easily available and it has gone through more tests, making it more stable), so if an official release is announced prior to your publication's final review, please consider updating to the official release.

The major version number is set by a major goal which is defined by the developers and user community before hand, for example see Section 1.5.1 [GNU Astronomy Utilities 1.0], page 6. The incremental work done in minor releases are commonly small steps in achieving the major goal. Therefore, there is no limit on the number of minor releases and the difference between the (hypothetical) versions 2.927 and 3.0 can be a very small (negligible to the user) improvement that finalizes the defined goals.

1.5.1 GNU Astronomy Utilities 1.0

Currently (prior to Gnuastro 1.0), the aim of Gnuastro is to have a complete system for data manipulation and analysis at least similar to IRAF[8]. So an astronomer can take all the standard data analysis steps (starting from raw data to the final reduced product and standard post-reduction tools) with the various programs in Gnuastro.

The maintainers of each camera or detector on a telescope can provide a completely transparent shell script or Makefile to the observer for data analysis. This script can set configuration files for all the required programs to work with that particular camera. The script can then run the proper programs in the proper sequence. The user/observer can easily follow the standard shell script to understand (and modify) each step and the parameters used easily. Bash (or other modern GNU/Linux shell scripts) are very powerful and made for this gluing job. This will simultaneously improve performance and transparency. Shell scripting (or Makefiles) are also very basic constructs that are easy to learn and readily available as part of the Unix-like operating systems. If there is no program to do a desired step, Gnuastro's libraries can be used to build specific programs.

[7] Each point in Gnuastro's history is uniquely identified with a 40 character long hash which is created from its contents and previous history for example: `5b17501d8f29ba3cd610673261e6e2229c846d35`. So the string `D` in the version for this commit could be `5b17`, or `5b175`.

[8] `http://iraf.noao.edu/`

The main factor is that all observatories or projects can freely contribute to Gnuastro and all simultaneously benefit from it (since it doesn't belong to any particular one of them), much like how for-profit organizations (for example RedHat, or Intel and many others) are major contributors to free and open source software for their shared benefit. Gnuastro's copyright has been fully awarded to GNU, so it doesn't belong to any particular astronomer or astronomical facility or project.

1.6 New to GNU/Linux?

Some astronomers initially install and use the GNU/Linux operating systems because the software that their research community use can only be run in this environment, the transition is not necessarily easy. To encourage you in investing the patience and time to make this transition, we define the GNU/Linux system and argue for the command-line interface of scientific software and how it is worth the (apparently steep) learning curve. Section 1.6.1 [Command-line interface], page 7, contains a short overview of the very powerful command-line user interface. Chapter 2 [Tutorials], page 14, is a complete chapter with some real world example applications of Gnuastro making good use of GNU/Linux capabilities written for newcomers to this environment. It is fully explained, easy and (hopefully) entertaining.

You might have already noticed that we are not using the name "Linux", but "GNU/Linux". Please take the time to have a look at the following essays and FAQs for a complete understanding of this very important distinction. In short, the Linux kernel is built using the GNU C library (glibc) and GNU compiler collection (gcc). The Linux kernel software alone is useless, in order have an operating system you need many more packages and the majority of such low-level packages in most distributions are developed as part of the GNU project: "the whole system is basically GNU with Linux loaded". In the form of an analogy: to say "running Linux", is like saying "driving your carburetor".

- `https://www.gnu.org/gnu/gnu-users-never-heard-of-gnu.html`
- `https://www.gnu.org/gnu/linux-and-gnu.html`
- `https://www.gnu.org/gnu/why-gnu-linux.html`
- `https://www.gnu.org/gnu/gnu-linux-faq.html`

1.6.1 Command-line interface

One aspect of Gnuastro that might be a little troubling to new GNU/Linux users is that (at least for the time being) it only has a command-line user interface (CLI). This might be contrary to the mostly graphical user interface (GUI) experience with proprietary operating systems. To a first time user, the command-line does appear much more complicated and adapting to it might not be easy and a little frustrating at first. This is understandable and also experienced by anyone who started using the computer (from childhood) in a graphical user interface. Here we hope to convince you of the unique benefits of this interface which can greatly enhance your productivity while complementing your GUI experience.

Through GNOME 3[9], most GNU/Linux based operating systems now have a very advanced and useful GUI. Since the GUI was created long after the command-line, some wrongly consider the command line to be obsolete. Both interfaces are very useful for different tasks (for example you can't view an image, video, pdf document or web page on

[9] `http://www.gnome.org/`

the command-line!), on the other hand you can't reproduce your results easily in the GUI. Therefore they should not be regarded as rivals but as complementary user interfaces, here we will outline how the CLI can be useful in scientific programs.

You can think of the GUI as a veneer over the CLI to facilitate a small subset of all the possible CLI operations. Each click you do on the GUI, can be thought of as internally running a different CLI command. So asymptotically (if a good designer can design a GUI which is able to show you all the possibilities to click on) the GUI is only as powerful as the command-line. In practice, such graphical designers are very hard to find for every program, so the GUI operations are always a subset of the internal CLI commands. For programs that are only made for the GUI, this results in not including lots of potentially useful operations. It also results in 'interface design' to be a crucially important part of any GUI program. Scientists don't usually have enough resources to hire a graphical designer, also the complexity of the GUI code is far more than CLI code, which is harmful for a scientific software, see Section 1.2 [Science and its tools], page 2.

For programs that have a GUI, one action on the GUI (moving and clicking a mouse, or tapping a touchscreen) might be more efficient and easier than its CLI counterpart (typing the program name and your desired configuration). However, if you have to repeat that same action more than once, the GUI will soon become very frustrating and prone to errors. Unless the designers of a particular program decided to design such a system for a particular GUI action, there is no general way to run any possible series of actions automatically on the GUI.

On the command-line, you can run any series of of actions which can come from various CLI capable programs you have decided your self in any possible permutation with one command[10]. This allows for much more creativity and exact reproducability that is not possible to a GUI user. For technical and scientific operations, where the same operation (using various programs) has to be done on a large set of data files, this is crucially important. It also allows exact reproducability which is a foundation principle for scientific results. The most common CLI (which is also known as a shell) in GNU/Linux is GNU Bash, we strongly encourage you to put aside several hours and go through this beautifully explained web page: `https://flossmanuals.net/command-line/`. You don't need to read or even fully understand the whole thing, only a general knowledge of the first few chapters are enough to get you going.

Since the operations in the GUI are very limited and they are visible, reading a manual is not that important in the GUI (most programs don't even have any!). However, to give you the creative power explained above, with a CLI program, it is best if you first read the manual of any program you are using. You don't need to memorize any details, only an understanding of the generalities is needed. Once you start working, there are more easier ways to remember a particular option or operation detail, see Section 4.6 [Getting help], page 57.

To experience the command-line in its full glory and not in the GUI terminal emulator, press the following keys together: `CTRL+ALT+F4`[11] to access the virtual console. To return

[10] By writing a shell script and running it, for example see the tutorials in Chapter 2 [Tutorials], page 14.

[11] Instead of `F4`, you can use any of the keys from `F1` to `F6` for different virtual consoles depending on your GNU/Linux distribution, try them all out. You can also run a separate GUI from within this console if you want to.

back to your GUI, press the same keys above replacing `F4` with `F7` (or `F1`, or `F2`, depending on your GNU/Linux distribution). In the virtual console, the GUI, with all its distracting colors and information, is gone. Enabling you to focus entirely on your actual work.

For operations that use a lot of your system's resources (processing a large number of large astronomical images for example), the virtual console is the place to run them. This is because the GUI is not competing with your research work for your system's RAM and CPU. Since the virtual consoles are completely independent, you can even log out of your GUI environment to give even more of your hardware resources to the programs you are running and thus reduce the operating time.

Since it uses far less system resources, the CLI is also very convenient for remote access to your computer. Using secure shell (SSH) you can log in securely to your system (similar to the virtual console) from anywhere even if the connection speeds are low. There are apps for smart phones and tablets which allow you to do this.

1.7 Report a bug

According to Wikipedia "a software bug is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways". So when you see that a program is crashing, not reading your input correctly, giving the wrong results, or not writing your output correctly, you have found a bug. In such cases, it is best if you report the bug to the developers. The programs will also report bugs in known impossible situations (which are caused by something unexpected) and will ask the users to report the bug.

Prior to actually filing a bug report, it is best to search previous reports. The issue might have already been found and even solved. The best place to check if your bug has already been discussed is the bugs tracker on Section 11.3 [Gnuastro project webpage], page 238, at `https://savannah.gnu.org/bugs/?group=gnuastro`. In the top search fields (under "Display Criteria") set the "Open/Closed" drop-down menu to "Any" and choose the respective program or general category of the bug in "Category" and click the "Apply" button. The results colored green have already been solved and the status of those colored in red is shown in the table.

Recently corrected bugs are probably not yet publicly released because they are scheduled for the next Gnuastro stable release. If the bug is solved but not yet released and it is an urgent issue for you, you can get the version controlled source and compile that, see Section 3.2.2 [Version controlled source], page 31.

To solve the issue as readily as possible, please follow the following to guidelines in your bug report. The How to Report Bugs Effectively (`http://www.chiark.greenend.org.uk/~sgtatham/bugs.html`) and How To Ask Questions The Smart Way (`http://catb.org/~esr/faqs/smart-questions.html`) essays also provide some very good generic advice for all software (don't contact their authors for Gnuastro's problems). Mastering the art of giving good bug reports (like asking good questions) can greatly enhance your experience with any free and open source software. So investing the time to read through these essays will greatly reduce your frustration after you see something doesn't work the way you feel it is supposed to for a large range of software, not just Gnuastro.

Be descriptive

Please provide as many details as possible and be very descriptive. Explain what you expected and what the output was: it might be that your expectation was wrong. Also please clearly state which sections of the Gnuastro book (this book), or other references you have studied to understand the problem. This can be useful in correcting the book (adding links to likely places where users will check). But more importantly, it will be very encouraging for the developers, since you are showing how serious you are about the problem and that you have actually put some thought into it. "To be able to ask a question clearly is two-thirds of the way to getting it answered." – John Ruskin (1819-1900).

Individual and independent bug reports

If you have found multiple bugs, please send them as separate (and independent) bugs (as much as possible). This will significantly help us in managing and resolving them sooner.

Reproducible bug reports

If we cannot exactly reproduce your bug, then it is very hard to resolve it. So please send us a Minimal working example[12] along with the description. For example in running a program, please send us the full command-line text and the output with the `-P` option, see Section 4.4 [Final parameter values, reproduce previous results], page 56. If it is caused only for a certain input, also send us that input file. In case the input FITS is large, please use ImageCrop to only crop the problematic section and make it as small as possible so it can easily be uploaded and downloaded and not waste the archive's storage, see Section 6.1 [ImageCrop], page 75.

There are generally two ways to inform us of bugs:

- Send a mail to `bug-gnuastro@gnu.org`. Any mail you send to this address will be distributed through the bug-gnuastro mailing list[13]. This is the simplest way to send us bug reports. The developers will then register the bug into the project webpage (next choice) for you.
- Use the Gnuastro project webpage at `https://savannah.gnu.org/projects/gnuastro/`: There are two ways to get to the submission page as listed below. Fill in the form as described below and submit it (see Section 11.3 [Gnuastro project webpage], page 238, for more on the project webpage).
  - Using the top horizontal menu items, immediately under the top page title. Hovering your mouse on "Support" will open a drop-down list. Select "Submit new".
  - In the main body of the page, under the "Communication tools" section, click on "Submit new item".

Once the items have been registered in the mailing list or webpage, the developers will add it to either the "Bug Tracker" or "Task Manager" trackers of the Gnuastro project webpage. These two trackers can only be edited by the Gnuastro project developers, but they can be browsed by anyone, so you can follow the progress on your bug. You are most

[12] `http://en.wikipedia.org/wiki/Minimal_Working_Example`

[13] `https://lists.gnu.org/mailman/listinfo/bug-gnuastro`

welcome to join us in developing Gnuastro and fixing the bug you have found maybe a good starting point. Gnuastro is designed to be easy for anyone to develop (see Section 1.2 [Science and its tools], page 2) and there is a full chapter devoted to developing it: Chapter 11 [Developing], page 236.

1.8 Suggest new feature

We would always be very happy to hear of suggested new features. For every program there are already lists of features that we are planning to add. You can see the current list of plans from the Gnuastro project webpage at `https://savannah.gnu.org/projects/gnuastro/` and following "Tasks"→"Browse" on the horizontal menu at the top of the page immediately under the title, see Section 11.3 [Gnuastro project webpage], page 238. If you want to request a feature to an existing program, click on the "Display Criteria" above the list and under "Category", choose that particular program. Under "Category" you can also see the existing suggestions for new programs or other cases like installation, documentation or libraries. Also be sure to set the "Open/Closed" value to "Any".

If the feature you want to suggest is not already listed in the task manager, then follow the steps that are fully described in Section 1.7 [Report a bug], page 9. Please have in mind that the developers are all very busy with their own astronomical research, and implementing existing "task"s to add or resolving bugs. Gnuastro is a volunteer effort and none of the developers are paid for their hard work. So, although we will try our best, please don't not expect that your suggested feature be immediately included (with the next release of Gnuastro).

The best person to apply the exciting new feature you have in mind is you, since you have the motivation and need. Infact Gnuastro is designed for making it as easy as possible for you to hack into it (add new features, change existing ones and so on), see Section 1.2 [Science and its tools], page 2. Please have a look at the chapter devoted to developing (Chapter 11 [Developing], page 236) and start applying your desired feature. Once you have added it, you can use it for your own work and if you feel you want others to benefit from your work, you can request for it to become part of Gnuastro. You can then join the developers and start maintaining your own part of Gnuastro. If you choose to take this path of action please contact us before hand (Section 1.7 [Report a bug], page 9) so we can avoid possible duplicate activities and get interested people in contact.

Gnuastro is a collection of low level programs: As described in Section 11.2 [Program design philosophy], page 238, a founding principle of Gnuastro is that each library or program should be very basic and low-level. High level jobs should be done by running the separate programs or using separate functions in succession through a shell script or calling the libraries by higher level functions, see the examples in Chapter 2 [Tutorials], page 14. So when making the suggestions please consider how your desired job can best be broken into separate steps and modularized.

1.9 Announcements

Gnuastro has a dedicated mailing list for making announcements. Anyone that is interested can subscribe to this mailing list to stay up to date with new releases or when the depen-

dencies (see Section 3.1 [Dependencies], page 24) have been updated. To subscribe to this list, please visit `https://lists.gnu.org/mailman/listinfo/info-gnuastro`.

1.10 Conventions

In this book we have the following conventions:

- All commands that are to be run on the shell (command-line) prompt as the user start with a `$`. In case they must be run as a super-user or system administrator, they will start with a `#`. If the command is in a separate line and next line `is also in the code type face`, but doesn't have any of the `$` or `#` signs, then it is the output of the command after it is run. As a user, you don't need to type those lines.
- If the command becomes larger than the page width a `\` is inserted in the code. If you are typing the code by hand on the command-line, you don't need to use multiple lines or add the extra space characters, so you can omit them. If you want to copy and paste these examples (highly discouraged!) then the `\` should stay.

  The `\` character is a shell escape character which is used commonly to make characters which have special meaning for the shell loose that special place (the shell will not treat them specially if there is a `\` behind them). When it is a last character in a line (the next character is a new-line character) the new-line character looses its meaning an the shell sees it as a simple white-space character, enabling you to use multiple lines to write your commands.

1.11 Acknowledgments

The list of Gnuastro authors is available at the start of this book and the `AUTHORS` file in the source code. Here the authors wish to gratefully acknowledge the help and support they received from other people and institutions who had an indirect (not committed in the version controlled history) role in Gnuastro. The plain text file `THANKS` which is distributed along with the source code also contains this list.

The Japanese Ministry of Science and Technology (MEXT) scholarship for Mohammad Akhlaghi's Masters and PhD period in Tohoku University Astronomical Institute had an instrumental role in the long term learning and planning that made the idea of Gnuastro possible. The very critical view points of Professor Takashi Ichikawa (from Tohoku University) were also instrumental in the initial ideas and creation of Gnuastro. Brandon Invergo, Karl Berry and Richard Stallman also provided very useful suggestions during the GNU evaluation process. Bob Proulx from Savannah, has kindly supported Gnuastro's project webpage on Savannah and the management of its version controlled source server there.

We would also like to gratefully thank Mohammad-reza Khellat, Alan Lefor, and Yahya Sefidbakht for their useful and constructive comments and suggestions. Finally we should thank all the (sometimes anonymous) developers in various online forums which patiently answered all our small (but important) technical questions. All work on Gnuastro has been voluntary, but we are most grateful to the following institutions (in chronological order) for hosting us in our research:

Ministry of education, culture, sports, science and technology (MEXT), Japan.
Tohoku University Astronomical Institute, Sendai, Japan.
University of Salento, Lecce, Italy.

Centre national de la recherche scientifique (CNRS), France.
Centre de Recherche Astrophysique de Lyon, University of Lyon 1, France.

2 Tutorials

In this chapter we give several tutorials or cookbooks on how to use the various tools in Gnuastro for your scientific purposes. In these tutorials, we have intentionally avoided too many cross references to make it more easily readable. To get more information about a particular program, you can visit the section with the same name as the program in this book. Each program section starts by explaining the general concepts behind what it does. If you only want to see an explanation of the options and arguments of any program, see the subsection titled 'Invoking ProgramName'. See Section 1.10 [Conventions], page 12, for an explanation of the conventions we use in the example codes through the book.

The tutorials in this section use a fictional setting of some historical figures in the history of astronomy. We have tried to show how Gnuastro would have been helpful for them in making their discoveries if there were GNU/Linux computers in their times! Please excuse us for any historical inaccuracy, this is not intended to be a historical reference. This form of presentation can make the tutorials more pleasant and entertaining to read while also being more practical (explaining from a user's point of view)[1]. The main reference for the historical facts mentioned in these fictional settings was Wikipedia.

2.1 Hubble visually checks and classifies his catalog

In 1924 Hubble[2] announced his discovery that some of the known nebulous objects are too distant to be within the the Milky Way (or Galaxy) and that they were probably distant Galaxies[3] in their own right. He had also used them to show that the redshift of the nebulae increases with their distance. So now he wants to study them more accurately to see what they actually are. Since they are nebulous or amorphous, they can't be modeled (like stars that are always a point) easily. So there is no better way to distinguish them than to visually inspect them and see if it is possible to classify these nebulae or not.

Hubble has stored all the FITS images of the objects he wants to visually inspect in his `/mnt/data/images` directory. He has also stored his catalog of extra Galactic nebulae in `/mnt/data/catalogs/extragalactic.txt`. Any normal user on his GNU/Linux system (including himself) only has read access to the contents of the `/mnt/data` directory. He has done this by running this command as root:

```
# chmod -R 755 /mnt/data
```

[1] This form of presenting a tutorial was influenced by the PGF/TikZ and Beamer manuals. The first provides graphic capabilities, while with the second you can make presentation slides in TEX and LATEX. In these manuals, Till Tantau (author of the manual) uses Euclid as the protagonist. There are also some nice words of wisdom for Unix-like systems called "Rootless Root": `http://catb.org/esr/writings/unix-koans/`. These also have a similar style but they use a mythical figure named Master Foo. If you already have some experience in Unix-like systems, you will definitely find these "Unix Koans" very entertaining.

[2] Edwin Powell Hubble (1889 – 1953 A.D.) was an American astronomer who can be considered as the father of extra-galactic astronomy, by proving that some nebulae are too distant to be within the Galaxy. He then went on to show that the universe appears to expand and also done a visual classification of the galaxies that is known as the Hubble fork.

[3] Note that at that time, "Galaxy" was a proper noun used to refer to the Milky way. The concept of a galaxy as we define it today had not yet become common. Hubble played a major role in creating today's concept of a galaxy.

Hubble has done this intentionally to avoid mistakenly deleting or modifying the valuable images he has taken at Mount Wilson while he is working as an ordinary user. Retaking all those images and data is simply not an option. In fact they are also in another hard disk (`/dev/sdb1`). So if the hard disk which stores his GNU/Linux distribution suddenly malfunctions due to work load, his data is not in harms way. That hard disk is only mounted to this directory when he wants to use it with the command:

```
# mount /dev/sdb1 /mnt/data
```

In short, Hubble wants to keep his data safe and fortunately by default Gnuastro allows for this. Hubble creates a temporary `visualcheck` directory in his home directory for this check. He runs the following commands to make the directory and change to it[4]:

```
$ mkdir ~/visualcheck
$ cd ~/visualcheck
$ pwd
/home/edwin/visualcheck
$ ls
```

Hubble has multiple images in `/mnt/data/images`, some of his targets might be on the edges of an image and so several images need to be stitched to give a good view of them. Also his extra Galactic targets belong to various pointings in the sky, so they are not in one large image. Gnuastro's ImageCrop is just the program he wants. The catalog in `extragalactic.txt` is a plain text file which stores the basic information of all his known 200 extra Galactic nebulae. In its second column it has each object's Right Ascension (the first column is a label he has given to each object) and in the third the object's declination. Having read the Gnuastro manual, he knows that all counting is done starting from zero, so the RA and Dec columns have number 1 and 2 respectively.

```
$ astimgcrop --racol=1 --deccol=2 /mnt/data/images/*.fits     \
             /mnt/data/catalogs/extragalactic.txt
ImageCrop started on Tue Jun  14 10:18:11 1932
  ---- ./4_crop.fits                  1 1
  ---- ./2_crop.fits                  1 1
  ---- ./1_crop.fits                  1 1
[[[ Truncated middle of list ]]]
  ---- ./198_crop.fits                1 1
  ---- ./195_crop.fits                1 1
  - 200 images created.
  - 200 were filled in the center.
  - 0 used more than one input.
ImageCrop finished in:  2.429401 (seconds)
```

Hubble already knows that thread allocation to the the CPU cores is asynchronous, so each time you run it the order of which job gets done first differs. When using ImageCrop the order of outputs is irrelevant since each crop is independent of the rest. This is why the crops are not necessarily created in the same input order. He is content with the default width of the outputs (which he inspected by running `$ astimgcrop -P`). If he wanted a different width for the cropped images, he could do that with the `--wwidth` option which

[4] The `pwd` command is short for "Print Working Directory" and `ls` is short for "list" which shows the contents of a directory.

accepts a value in arcseconds. When he lists the contents of the directory again he finds his 200 objects as separate FITS images.

```
$ ls
1_crop.fits 2_crop.fits ... 200_crop.fits
```

The FITS image format was not designed for viewing, but mainly for accurate storing of the data. So he chooses to convert the cropped images to a more common image format to view them more quickly and easily through standard image viewers (which load much faster than FITS image viewer). JPEG is one of the most recognized image formats that is supported by most image viewers. Fortunately Gnuastro has just such a tool to convert various types of file types to and from each other: ConvertType. Hubble has already heard of GNU Parallel from one of his colleagues at Mount Wilson Observatory. It allows multiple instances of a command to be run simultaneously on the system, so he uses it in conjunction with ConvertType to convert all the images to JPEG.

```
$ parallel astconvertt -ojpg ::: *_crop.fits
```

For his graphical user interface Hubble is using GNOME which is the default in most distributions in GNU/Linux. The basic image viewer in GNOME is the Eye of GNOME, which has the executable file name `eog`[5]. Since he has used it before, he knows that once it opens an image, he can use the `ENTER` or `SPACE` keys on the keyboard to go to the next image in the directory or the `Backspace` key to to go the previous image. So he opens the image of the first object with the command below and with his cup of coffee in his other hand, he flips through his targets very fast to get a good initial impression of the morphologies of these extra Galactic nebulae.

```
$ eog 1_crop.jpg
```

Hubble's cup of coffee is now finished and he also got a nice general impression of the shapes of the nebulae. He tentatively/mentally classified the objects into three classes while doing the visual inspection. One group of the nebulae have a very simple elliptical shape and seem to have no internal special structure, so he gives them code 1. Another clearly different class are those which have spiral arms which he associates with code 2 and finally there seems to be a class of nebulae in between which appear to have a disk but no spiral arms, he gives them code 3.

Now he wants to know how many of the nebulae in his extra Galactic sample are within each class. Repeating the same process above and writing the results on paper is very time consuming and prone to errors. Fortunately Hubble knows the basics of GNU Bash shell programming, so he writes the following short script with a loop to help him with the job. After all, computers are made for us to operate and knowing basic shell programming gives Hubble this ability to creatively operate the computer as he wants. So using GNU Emacs[6] (his favorite text editor) he puts the following text in a file named `classify.sh`.

```
for name in *.jpg
do
    eog $name &
    processid=$!
```

[5] Eye of GNOME is only available for users of the GNOME graphical desktop environment which is the default in most GNU/Linux distributions. If you use another graphical desktop environment, replace `eog` with any other image viewer.

[6] This can be done with any text editor

```
    echo -n "$name belongs to class: "
    read class
    echo $name $class >> classified.txt
    kill $processid
done
```

Fortunately GNU Emacs or even simpler editors like Gedit (part of the GNOME graphical user interface) will display the variables and shell constructs in different colors which can really help in understanding the script. Put simply, the `for` loop gets the name of each JPEG file in the directory this script is run in and puts it in `name`. In the shell, the value of a variable is used by putting a `$` sign before the variable name. Then Eye of GNOME is run on the image in the background to show him that image and its process ID is saved internally (this is necessary to close Eye of GNOME later). The shell then prompts the user to specify a class and after saving it in `class`, it prints the file name and the given class in the next line of a file named `classified.txt`. To make the script executable (so he can run it later any time he wants) he runs:

```
$ chmod +x classify.sh
```

Now he is ready to do the classification, so he runs the script:

```
$ ./classify.sh
```

In the end he can delete all the JPEG and FITS files along with ImageCrop's log file with the following short command. The only files remaining are the script and the result of the classification.

```
$ rm *.jpg *.fits astimgcrop.txt
$ ls
classified.txt   classify.sh
```

He can now use `classified.txt` as input to a plotting program to plot the histogram of the classes and start making interpretations about what these nebulous objects that are outside of the Galaxy are.

2.2 Sufi simulates a detection

It is the year 953 A.D. and Sufi[7] is in Shiraz as a guest astronomer. He had come there to use the advanced 123 centimeter astrolabe for his studies on the Ecliptic. However, something was bothering him for a long time. While mapping the constellations, there were several non-stellar objects that he had detected in the sky, one of them was in the Andromeda constellation. During a trip he had to Yemen, Sufi had seen another such object in the southern skies looking over the Indian ocean. He wasn't sure if such cloud-like non-stellar objects (which he was the first to call 'Sahābi' in Arabic or 'nebulous') were real astronomical objects or if they were only the result of some bias in his observations. Could such diffuse objects actually be detected at all with his detection technique?

He still had a few hours left until nightfall (when he would continue his studies on the ecliptic) so he decided to find an answer to this question. He had thoroughly studied Claudius Ptolemy's (90 – 168 A.D) Almagest and had made lots of corrections to it, in

[7] Abd al-rahman Sufi (903 – 986 A.D.), also known in Latin as Azophi was an Iranian astronomer. His manuscript "Book of fixed stars" contains the first recorded observations of the Andromeda galaxy, the Large Magellanic Cloud and seven other non-stellar or 'nebulous' objects.

particular in measuring the brightness. Using his same experience, he was able to measure a magnitude for the objects and wanted to simulate his observation to see if a simulated object with the same brightness and size could be detected in a simulated noise with the same detection technique. The general outline of the steps he wants to take are:

1. Make some mock profiles in an oversampled image. The initial mock image has to be oversampled prior to convolution or other forms of transformation in the image. Through his experiences, Sufi knew that this is because the image of heavenly bodies is actually transformed by the atmosphere or other sources outside the atmosphere (for example gravitational lenses) prior to being sampled on an image. Since that transformation occurs on a continuous grid, to best approximate it, he should do all the work on a finer pixel grid. In the end he can resample the result to the initially desired grid size.
2. Convolve the image with a PSF image that is oversampled to the same value as the mock image. Since he wants to finish in a reasonable time and the PSF kernel will be very large due to oversampling, he has to use frequency domain convolution which has the side effect of dimming the edges of the image. So in the first step above he also has to build the image to be larger by at least half the width of the PSF convolution kernel on each edge.
3. With all the transformations complete, the image should be resampled to the same size of the pixels in his detector.
4. He should remove those extra pixels on all edges to remove frequency domain convolution artifacts in the final product.
5. He should add noise to the (until now, noise-less) mock image. After all, all observations have noise associated with them.

Fortunately Sufi had heard of GNU Astronomy Utilities from a colleague in Isfahan (where he worked) and had installed it on his computer a year before. It had tools to do all the steps above. He had used MakeProfiles before, but wasn't sure which columns he had chosen in his user or system wide configuration files for which parameters, see Section 4.2 [Configuration files], page 51. So to start his simulation, Sufi runs MakeProfiles with the `-P` option to make sure what columns in a catalog MakeProfiles currently recognizes and the output image parameters:

```
$ astmkprof -P
# MakeProfiles (GNU Astronomy Utilities 0.1) 0.1
# Configured on 21 September 952 at 19:37
# Written on Sat Oct  6 15:49:31 953

# Output:
 naxis1              1000
 naxis2              1000
 oversample          5

[[[ Truncated middle of list ]]]

# Catalog:
 xcol                1
```

```
ycol              2
fcol              3
rcol              4
ncol              5
pcol              6
qcol              7
mcol              8
tcol              9

[[[ Truncated rest of list ]]]
```

In particular, Sufi looks at the parameters under the catalog grouping. Fortunately the columns are naturally numbered such that column 0 can be an ID he specifies for each object (which MakeProfiles ignores) and each subsequent column specifies a given parameter. Fortunately MakeProfiles has the capability to also make the PSF which is to be used on the mock image and using the `--prepforconv` option, he can also make the mock image to be larger by the correct amount and all the sources to be shifted by the correct amount.

For his initial check he decides to simulate the nebula in the Andromeda constellation. The night he was observing, the PSF had roughly a FWHM of about 5 pixels, so as the first row, he defines the PSF parameters and sets the radius column (`rcol` above, fifth column) to `5.000`, he also chooses a Moffat function for its functional form. Remembering how diffuse the nebula in the Andromeda constellation was, he decides to simulate it with a mock Sérsic index 1.0 profile. He wants the output to be 500 pixels by 500 pixels, so he puts the mock profile in the center. Looking at his drawings of it, he decides a reasonable effective radius for it would be 40 pixels on this image pixel scale, he sets the axis ratio and position angle to approximately correct values too and finally he sets the total magnitude of the profile to 3.44 which he had accurately measured. Sufi also decides to truncate both the mock profile and PSF at 5 times the respective radius parameters. In the end he decides to put four stars on the four corners of the image at very low magnitudes as a visual scale.

Using all the information above, he creates the catalog of mock profiles he wants in a file named `cat.txt` (short for catalog) using his favorite text editor and stores it in a directory named `simulationtest` in his home directory. [The `cat` command prints the contents of a file, short for concatenation. So please copy-paste the outputs of the command "`cat cat.txt`" into `cat.txt` when the editor opens in the steps above it, note that there are 6 lines]:

```
$ mkdir ~/simulationtest
$ cd ~/simulationtest
$ pwd
/home/rahman/simulationtest
$ emacs cat.txt
$ ls
cat.txt
$ cat cat.txt
 0  0.0000   0.0000  1  5.000  4.765  0.0000  1.000  30.000  5.000
 1  250.00   250.00  0  40.00  1.000  -25.00  0.400  3.4400  5.000
 2  50.000   50.000  3  0.000  0.000  0.0000  0.000  9.0000  0.000
 3  450.00   50.000  3  0.000  0.000  0.0000  0.000  9.2500  0.000
```

```
 4  50.000   450.00  3  0.000  0.000  0.0000  0.000  9.5000  0.000
 5  450.00   450.00  3  0.000  0.000  0.0000  0.000  9.7500  0.000
```

The zero-point magnitude for his observation was 18. Now he has all the necessary parameters and runs MakeProfiles with the following command:

```
$ astmkprof --prepforconv --naxis1=500 --naxis2=500         \
            --zeropoint=18.0 cat.txt
MakeProfiles started on Sat Oct  6 16:26:56 953
  - 6 profiles read from cat.txt             in 0.000209 seconds
  ---- Row 5 complete, 5 left to go.
  ---- Row 3 complete, 4 left to go.
  ---- Row 2 complete, 3 left to go.
  ---- Row 4 complete, 2 left to go.
  ---- ./0_cat.fits created.
  ---- Row 0 complete, 1 left to go.
  ---- Row 1 complete, 0 left to go.
  - cat.fits created.                        in 0.024811 seconds
MakeProfiles finished in:  0.236629 (seconds)
$ls
0_cat.fits  astmkprof.log  cat.fits  cat.txt
```

The file `0.fits` is the PSF Sufi had asked for and `cat.fits` is the image containing the 5 objects. The PSF is now available to him as a separate file for the convolution step. While he was preparing the catalog, one of his students came up and was also following the steps. When he opened the image, the student was surprised to see that all the stars are only one pixel and not in the shape of the PSF as we see when we image the sky at night. So Sufi explained to him that the stars will take the shape of the PSF after convolution and this is how they would look if we didn't have an atmosphere or an aperture when we took the image. The size of the image was also surprising for the student, instead of 500 by 500, it was 2630 by 2630 pixels. So Sufi had to explain why oversampling is very important for parts of the image where the flux change is significant over a pixel. Sufi then explained to him that after convolving we will resample the image to get our originally desired size. To convolve the image, Sufi ran the following command:

```
$ astconvolve --kernel=0_cat.fits cat.fits
Convolve started on Mon Apr  6 16:35:32 953
Convolving cat.fits (hdu: 0)
 with the kernel 0.fits (hdu: 0).
 using 8 CPU threads in the frequency domain.
  - Input and Kernel images padded.          in 0.045576 seconds
  - Images converted to frequency domain.    in 10.486712 seconds
  - Multiplied in the frequency domain.      in 0.032780 seconds
  - Converted back to the spatial domain.    in 5.342335 seconds
  - Padded parts removed.                    in 0.011880 seconds
Convolve finished in:  15.972771 (seconds)
$ls
0_cat.fits  astmkprof.log  cat_convolved.fits  cat.fits  cat.txt
```

When convolution finished, Sufi opened the `cat_convolved.fits` file and showed the effect of convolution to his student and explained to him how a PSF with a larger FWHM would make the points even wider. With the convolved image ready, they were ready to re-sample it to the original pixel scale Sufi had planned. Sufi explained the basic concepts of warping the image to his student and also the fact that since the center of a pixel is assumed to take integer values in the FITS standard, the transformation matrix would not be a simple scaling but would also need translating, see Section 6.4.2 [Merging multiple warpings], page 112. Then he ran ImageWarp with the following command:

```
$ astimgwarp cat_convolved.fits --matrix="0.2,0,0.4  0,0.2,0.4  0,0,1"
ImageWarp started on Mon Apr  6 16:51:59 953
ImageWarp finished in:  0.481421 (seconds)
$ ls
0_cat.fits        cat_convolved.fits         cat.fits
astmkprof.log  cat_convolved_warped.fits  cat.txt
```

`cat_convolved_warped.fits` now has the correct pixel scale. However, the image is still larger than what we had wanted, it is 526 $(500 + 13 + 13)$ by 526 pixels. The student is slightly confused, so Sufi also resamples the PSF with ImageWarp and the same warping matrix and shows him that it is 27 $(2 \times 13 + 1)$ by 27 pixels. Sufi goes on to explain how frequency space convolution will dim the edges and that is why he added the `--prepforconv` option to MakeProfiles, see Section 8.1.2 [If convolving afterwards], page 167. Now that convolution is done Sufi can remove those extra pixels using ImageCrop:

```
$ astimgcrop cat_convolved_warped.fits --section=13:*-13,13:*-13  \
             --zeroisnotblank
ImageCrop started on Sat Oct  6 17:03:24 953
  - Read metadata of 1 images.               in 0.000560 seconds
  ---- cat_convolved_warped_crop.fits 1 1
ImageCrop finished in:  0.018917 (seconds)
$ls
0_cat.fits      astmkprof.log           cat_convolved_warped.fits
0_warped.fits   cat_convolved.fits              cat.fits
astimgcrop.log  cat_convolved_warped_crop.fits  cat.txt
```

Finally, the `cat_convolved_warped.fits` has the same dimensions as Sufi had asked for in the beginning. All this trouble was certainly worth it because now there is no dimming on the edges of the image and the profile centers are more accurately sampled. The final step to simulate a real observation would be to add noise to the image. Sufi set the zeropoint magnitude to the same value that he set when making the mock profiles and looking again at his observation log, he found that at that night the background flux near the nebula had a magnitude of 7. So using these values he ran MakeNoise:

```
$ astmknoise --zeropoint=18 --background=7 --output=out.fits    \
             cat_convolved_warped_crop.fits
MakeNoise started on Mon Apr  6 17:05:06 953
  - Generator type: mt19937
  - Generator seed: 1428318100
MakeNoise finished in:  0.033491 (seconds)
$ls
0_cat.fits      cat_convolved.fits              cat.txt
```

```
0_warped.fits   cat_convolved_warped_crop.fits  out.fits
astimgcrop.log  cat_convolved_warped.fits
astmkprof.log   cat.fits
```

The `out.fits` file now has the noised image of the mock catalog Sufi had asked for. Seeing how the `--output` option allows the user to specify the name of the output file, the student was confused and wanted to know why Sufi hadn't used it before? Sufi then explained to him that for intermediate steps it is best to rely on the automatic output, see Section 4.5 [Automatic output], page 57. Doing so will give all the intermediate files the same basic name structure, so in the end you can simply remove them all with the Shell's capabilities. So Sufi decided to show this to the student by making a shell script from the commands he had used before.

The command-line shell has the capability to read all the separate input commands from a file. This is very useful when you want to do the same thing multiple times, with only the names of the files or minor parameters changing between the different instances. Using the shell's history (by pressing the up keyboard key) Sufi reviewed all the commands and then he retrieved the last 5 commands with the `$ history 5` command. He selected all those lines he had input and put them in a text file named `mymock.sh`. Then he used some shell variables to set the two main constant parts of all the command to generalized variables.

```
edge=13
base=cat
rm out.fits

astmkprof --prepforconv --naxis1=500 --naxis2=500           \
          --zeropoint=18.0 "$base".txt
astconvolve --kernel=0.fits "$base".fits
astimgwarp "$base"_convolved.fits --matrix="0.2,0,0.4  0,0.2,0.4  0,0,1"
astimgcrop "$base"_convolved_warped.fits                    \
           --section=$edge:*-$edge,$edge:*-$edge
astmknoise --zeropoint=18 --background=7 --output=out.fits   \
           "$base"_convolved_warped_crop.fits
rm 0*.fits cat*.fits *.log
```

Sufi then explained to the eager student that you define a variable by giving it a name, followed by an `=` sign and the value you want. Then you can reference that variable from anywhere in the script by calling its name with a `$` prefix. So in the script whenever you see `$base`, the value we defined for it above is used. If you use advanced editors like GNU Emacs or even simpler ones like Gedit (part of the GNOME graphical user interface) the variables will become a different color which can really help in understanding the script. We have put all the `$base` variables in double quotation marks (`"`) so the variable name and the following text do not get mixed, the shell is going to ignore the `"` after replacing the variable value. To make the script executable, Sufi ran the following command:

```
$ chmod +x mymock.sh
```

Then finally, Sufi ran the script, simply by calling its file name:

```
$ ./mymock.sh
```

After the script finished, the only file remaining is the `out.fits` file that Sufi had wanted in the beginning. Sufi then explained to the student how he could run this script anywhere

that he has a catalog if the script is in the same directory. The only thing the student had to modify in the script was the name of the catalog (the value of the `base` variable in the start of the script) and the value to the `edge` variable if he changed the PSF size. The student was also very happy to hear that he won't need to make it executable again when he makes changes later, it will remain executable unless he explicitly changes the executable flag with `chmod`.

The student was really excited, since now, through simple shell scripting, he could really speed up his work and run any command in any fashion he likes allowing him to be much more creative in his works. Until now he was using the graphical user interface which doesn't have such a facility and doing repetitive things on it was really frustrating and some times he would make mistakes. So he left to go and try scripting on his own computer.

Sufi could now get back to his own work and see if the simulated nebula which resembled the one in the Andromeda constellation could be detected or not. Although it was extremely faint[8], fortunately it passed his detection tests and he wrote it in the draft manuscript that would later become "Book of fixed stars". He still had to check the other nebula he saw from Yemen and several other such objects, but they could wait until tomorrow (thanks to the shell script, he only has to define a new catalog). It was nearly sunset and they had to begin preparing for the night's measurements on the ecliptic.

[8] The brightness of a diffuse object is added over all its pixels to give its final magnitude, see Section 8.1.3 [Flux Brightness and magnitude], page 167. So although the magnitude 3.44 (of the mock nebula) is orders of magnitude brighter than 6 (of the stars), the central galaxy is much fainter. Put another way, the brightness is distributed over a large area in the case of a nebula.

3 Installation

To successfully install and thus use Gnuastro, first please make sure you have the dependencies installed (see Section 3.1 [Dependencies], page 24). Only the first group are required when you are building Gnuastro using a tarball (see Section 3.2.1 [Release tarball], page 30), they are very basic and low-level tools used in most astronomical software, so you might already have them installed, if not they are very easy to install as described for each. Section 3.2 [Downloading the source], page 30, discusses the two methods you can obtain the source code: as a tarball (a significant snapshot in Gnuastro's history), or the full history.

The building and installation of Gnuastro is heavily customizable, to learn more about them, see Section 3.3 [Build and install], page 34. This section is essentially a thorough explanation of the steps in Section 1.1 [Quick start], page 1. It discusses ways you can influence the building and installation. If you encounter any problems in the installation process, it is probably already explained in Section 3.3.4 [Known issues], page 43. In Appendix B [Other useful software], page 258, the installation and usage of some other free software that are not directly required by Gnuastro but might be useful in conjunction with it is discussed.

3.1 Dependencies

The dependencies needed to build and install Gnuastro are defined by the features you want and how you would like to obtain the source code (see Section 3.2 [Downloading the source], page 30). A minimal set of dependencies are mandatory, if they are not present you cannot get passed the configuration step. Such mandatory dependencies are therefore very basic (low-level) tools which are easy to obtain, build and install, see Section 3.1.1 [Mandatory dependencies], page 24, for a full discussion.

If you have the packages of Section 3.1.2 [Optional dependencies], page 27, Gnuastro will have additional functionality (for example converting FITS images to JPEG or PDF). If you are installing from a tarball as explained in Section 1.1 [Quick start], page 1, you can stop reading after this section. However, if you decided to use the version controlled source instead of the tarball (see Section 3.2.2 [Version controlled source], page 31), an additional bootstrapping step is required before configuration and its dependencies are explained in Section 3.1.3 [Bootstrapping dependencies], page 27.

3.1.1 Mandatory dependencies

The mandatory Gnuastro dependencies are very basic and low-level tools. They all follow the same basic GNU based build system (like that shown in Section 1.1 [Quick start], page 1), so even if you don't have them, installing them should be pretty straightforward. In this section we explain each program and any specific note that might be necessary in the installation.

The most basic choice is to build the packages from source yourself, instead of relying on your distribution's package management system. While the latter choice is indeed possible, we recommend that you build these dependencies yourself as discussed below. We will send out notifications in the `info-gnuastro` mailing list, see Section 1.9 [Announcements], page 11, when we find out that these requirements are updated.

1. For each package, Gnuastro might preform better (or require) certain configuration options that your distribution's package managers didn't add for you. If present, these configuration options are explained during the installation of each in the sections below. When the proper configuration has not been set, the programs should complain and inform you.
2. Your distribution's pre-built package might not be the most recent release.
3. For the libraries, they might separate the binary file from the header files, see Section 3.3.4 [Known issues], page 43.
4. Like any other tool, the science you derive from Gnuastro's tools highly depend on these lower level dependencies, so generally it is much better to have a close connection with them. By reading their manuals, installing them and staying up to date with changes/bugs in them, your scientific results and understanding will also correspondingly improve.

3.1.1.1 GNU Scientific library

The GNU Scientific Library (`http://www.gnu.org/software/gsl/`), or GSL, is a large collection of functions that are very useful in scientific applications, for example integration, random number generation, and Fast Fourier Transform among many others. To install GSL from source, you can run the following commands after you have downloaded `gsl-latest.tar.gz` (`ftp://ftp.gnu.org/gnu/gsl/gsl-latest.tar.gz`):

```
$ tar xf gsl-latest.tar.gz
$ cd gsl-X.X                    # Replace X.X with version number
$ ./configure
$ make
$ make check
$ sudo make install
```

3.1.1.2 CFITSIO

CFITSIO (`http://heasarc.gsfc.nasa.gov/fitsio/`) is the closest you can get to the pixels in a FITS image while remaining faithful to the FITS standard (`http://fits.gsfc.nasa.gov/fits_standard.html`). It is written by William Pence, the principal author of the FITS standard[1], and is regularly updated. Setting the definitions for all other software packages using FITS images.

Some GNU/Linux distributions have CFITSIO in their package managers, if it is available and updated, you can use it. One problem that might occur is that CFITSIO might not be configured with the `--enable-reentrant` option by the distribution. This option allows CFITSIO to open a file in multiple threads, it can thus provide great speed improvements. If CFITSIO was not configured with this option, any program which needs this capability will warn you and abort when you ask for multiple threads (see Section 4.3 [Threads in Gnuastro], page 54).

To install CFITSIO from source, we strongly recommend that you have a look through Chapter 2 (Creating the CFITSIO library) of the CFITSIO manual and understand the options you can pass to `$ ./configure` (they aren't too much). This is a very basic package

[1] Pence, W.D. et al. Definition of the Flexible Image Transport System (FITS), version 3.0. (2010) Astronomy and Astrophysics, Volume 524, id.A42, 40 pp.

for most astronomical software and it is best that you configure it nicely with your system. Once you download the source and unpack it, the following configure script should be enough for most purposes. Don't forget to read chapter two of the manual though, for example the second option is only for 64bit systems. The manual also explains how to check if it has been installed correctly.

CFITSIO comes with two executables called fpack and funpack. From their manual: they "are standalone programs for compressing and uncompressing images and tables that are stored in the FITS (Flexible Image Transport System) data format. They are analogous to the gzip and gunzip compression programs except that they are optimized for the types of astronomical images that are often stored in FITS format". The instruction below explain how to compile and install them on your system along with CFITSIO. They are not essential for Gnuastro, since they are just wrappers for functions within CFITSIO, but they can come in handy. The `make utils` command is only available for versions above 3.39, it will build these executables along with several other test executables which are deleted before the installation (otherwise they will also be installed).

The CFITSIO installation from source process is given below. Let's assume you have downloaded `cfitsio_latest.tar.gz` (`http://heasarc.gsfc.nasa.gov/FTP/software/fitsio/c/cfitsio_latest.tar.gz`) and are in the same directory:

```
$ tar xf cfitsio_latest.tar.gz
$ cd cfitsio
$ ./configure --prefix=/usr/local --enable-sse2 --enable-reentrant
$ make
$ make utils
$ ./testprog > testprog.lis
$ diff testprog.lis testprog.out    # Should have no output
$ cmp testprog.fit testprog.std     # Should have no output
$ rm cookbook fitscopy imcopy smem speed testprog
$ sudo make install
```

3.1.1.3 WCSLIB

WCSLIB (`http://www.atnf.csiro.au/people/mcalabre/WCS/`) is written and maintained by one of the authors of the World Coordinate System (WCS) definition in the FITS standard (`http://fits.gsfc.nasa.gov/fits_standard.html`)[2], Mark Calabretta. It might be already built and ready in your distribution's package management system. However, here the installation from source is explained, for the advantages please see Section 3.1.1 [Mandatory dependencies], page 24. To install WCSLIB you will need to have CFITSIO already installed, see Section 3.1.1.2 [CFITSIO], page 25.

WCSLIB also has plotting capabilities which use PGPLOT (a plotting library for C). If you wan to use those capabilities in WCSLIB, Section B.2 [PGPLOT], page 259, provides the PGPLOT installation instructions. However PGPLOT is old[3], so its installation is not easy, there are also many great modern WCS plotting tools (mostly in written in Python). Hence, if you will not be using those plotting functions in WCSLIB, you can configure it

[2] Greisen E.W., Calabretta M.R. (2002) Representation of world coordinates in FITS. Astronomy and Astrophysics, 395, 1061-1075.

[3] As of early June 2016, its most recent version was uploaded in February 2001.

with the `--without-pgplot` option as shown below. Let's assume you have downloaded `wcslib.tar.bz2` (`ftp://ftp.atnf.csiro.au/pub/software/wcslib/wcslib.tar.bz2`) and are in the same directory:

```
$ tar xf wcslib.tar.bz2
$ cd wcslib-X.X                     # Replace X.X with version number
$ ./configure --without-pgplot LIBS="-pthread -lm"
$ make
$ make check
$ sudo make install
```

3.1.2 Optional dependencies

The libraries listed here are only used for very specific applications, therefore if you don't want these operations, Gnuastro will be built and installed without them and you don't have to have the dependencies.

If the `./configure` script can't find these requirements, it will warn you in the end that they are not present and notify you of the operation(s) you can't do due to not having them. If the output you request from a program requires a missing library, that program is going to warn you and abort. In the case of executables like GPL GhostScript, if you install them at a later time, the program will run. This is because if required libraries are not present at build time, the executables cannot be built, but an executable is called by the built program at run time so if it becomes available, it will be used. If you do install an optional library later, you will have to rebuild Gnuastro and reinstall it for it to take effect.

libjpeg
: libjpeg is only used by ConvertType to read from and write to JPEG images. libjpeg (`http://www.ijg.org/`) is a very basic library that provides tools to read and write JPEG images, most of the GNU/Linux graphic programs and libraries use it. Therefore you most probably already have it installed. libjpeg-turbo (`http://libjpeg-turbo.virtualgl.org/`) is an alternative to libjpeg. It uses SIMD instructions for ARM based systems that significantly decreases the processing time of JPEG compression and decompression algorithms.

GPL Ghostscript
: GPL Ghostscript's executable (`gs`) is called used by ConvertType to compile a PDF file from a source PostScript file, see Section 5.2 [ConvertType], page 65. Therefore its headers (and libraries) are not needed. With a very high probability you already have it in your GNU/Linux distribution. Unfortunately it does not follow the standard GNU build style so installing it is very hard. It is best to rely on your distribution's package managers for this.

3.1.3 Bootstrapping dependencies

Bootstrapping is only necessary if you have decided to obtain the full version controlled history of Gnuastro, see Section 3.2.2 [Version controlled source], page 31, and Section 3.2.2.1 [Bootstrapping], page 32. Using the version controlled source enables you to always be up to date with the most recent development work of Gnuastro (bug fixes, new functionalities, improved algorithms and etc). If you have downloaded a tarball (see Section 3.2 [Downloading the source], page 30), then you can ignore this subsection.

To successfully run the bootstrapping process, there are some additional dependencies to those discussed in the previous subsections. These are low level tools that are used by a large collection of Unix-like operating systems programs, therefore they are most probably already available in your system. If they are not already installed, you should be able to easily find them in any GNU/Linux distribution package management system (`apt-get`, `yum`, `pacman` and etc). The short names in parenthesis in `typewriter` font after the package name can be used to search for them in your package manager. For the GNU Portability Library, GNU Autoconf Archive and TEX Live, it is recommended to use the instructions here, not your operating system's package manager.

GNU Portability Library (Gnulib)

To ensure portability for a wider range of operating systems (those that don't include GNU C library, namely glibc), Gnuastro depends on the GNU portability library, or Gnulib. Gnulib keeps a copy of all the functions in glibc, implemented (as much as possible) to be portable to other operating systems. The `bootstrap` script can automatically clone Gnulib (as a `gnulib/` directory inside Gnuastro), however, as described in Section 3.2.2.1 [Bootstrapping], page 32, this is not recommended.

The recommended way to bootstrap Gnuastro is to first clone Gnulib and the Autoconf archives (see below) into a local directory outside of Gnuastro. Let's call it `DEVDIR`[4] (which you can set to any directory). Currently in Gnuastro, both Gnulib and Autoconf archives have to be cloned in the same top directory[5] like the case here[6]:

```
$ DEVDIR=/home/yourname/Development
$ cd $DEVDIR
$ git clone git://git.sv.gnu.org/gnulib.git
$ git clone git://git.sv.gnu.org/autoconf-archive.git
```

You now have the full version controlled source of these two repositories in separate directories. Both these packages are regularly updated, so every once in a while, you can run `$ git pull` within them to get any possible updates.

GNU Automake (`automake`)

GNU Automake will build the `Makefile.in` files in each sub-directory using the (hand-written) `Makefile.am` files. The `Makefile.in`s are subsequently used to generate the `Makefile`s when the user runs `./configure` before building.

[4] If you are not a developer in Gnulib or Autoconf archives, `DEVDIR` can be a directory that you don't backup. In this way the large number of files in these projects won't slow down your backup process or take bandwidth (if you backup to a remote server).

[5] If you already have the Autoconf archives in a separate directory, or can't clone it in the same directory as Gnulib, or you have it with another directory name (not `autoconf-archive/`), you can follow this short step. Set `AUTOCONFARCHIVES` to your desired address. Then define a symbolic link in `DEVDIR` with the following command so Gnuastro's bootstrap script can find it:
`$ ln -s $AUTOCONFARCHIVES $DEVDIR/autoconf-archive`.

[6] If your internet connection is active, but Git complains about the network, it might be due to your network setup not recognizing the git protocol. In that case use the following URL for the HTTP protocol instead (for Autoconf archives, replace the name): `http://git.sv.gnu.org/r/gnulib.git`

GNU Autoconf (`autoconf`)
: GNU Autoconf will build the `configure` script using the configurations we have defined (hand-written) in `configure.ac`.

GNU Autoconf Archive
: These are a large collection of tests that can be called to run at `./configure` time. See the explanation under GNU Portability Library above for instructions on obtaining it and keeping it up to date.

GNU Libtool (`libtool`)
: GNU Libtool is in charge of building all the libraries in Gnuastro. The libraries contain functions that are used by more than one program and are installed for use in other programs. They are thus put in a separate directory (`lib/`).

GNU help2man (`help2man`)
: GNU help2man is used to convert the output of the `--help` option (Section 4.6.2 [`--help`], page 58) to the traditional Man page (Section 4.6.3 [Man pages], page 59).

LaTeX and some TeX packages
: Some of the figures in this book are built by LaTeX (using the PGF/TikZ package). The LaTeX source for those figures is version controlled for easy maintenance not the actual figures. So the `./boostrap` script will run LaTeX to build the figtures. The best way to install LaTeX and all the necessary packages is through TeX live (`https://www.tug.org/texlive/`) which is a package manager for TeX related tools that is independent of any operating system. It is thus preferred to the TeX Live versions distributed by your operating system.

  To install TeX Live, go to the webpage and download the appropriate installer by following the "download" link. Note that by default the full package repository will be downloaded and installed (around 4 Giga Bytes) which can take *very* long to download and to update later. However, most packages are not needed by everyone, it is easier, faster and better to install only the "Basic scheme" (consisting of only the most basic TeX and LaTeX packages, which is less than 200 Mega bytes)[7].

  After the installation be sure to set the environment variables as suggested in the end of the outputs. Any time you confront (need) a package you don't have, simply install it with a command like below (similar to how you install software from your operating system's package manager)[8]. To install all the necessary TeX packages for a successful Gnuastro bootstrap, run this command:

```
$ su
# tlmgr install epsf jknapltx caption biblatex biber iftex \
                etoolbox logreq xstring xkeyval pgf ms     \
                xcolor pgfplots times rsfs pstools epspdf
```

[7] You can also download the DVD iso file at a later time to keep as a backup for when you don't have internet connection if you need a package.

[8] After running TeX, or LaTeX, you might get a warning complaining about a `missingfile`. Run '`tlmgr info missingfile`' to see the package(s) containing that file which you can install.

ImageMagick (`imagemagick`)
: ImageMagick is a wonderful and robust program for image manipulation on the command-line. `bootsrap` uses it to convert the book images into the formats necessary for the various book formats.

3.2 Downloading the source

Gnuastro's source code can be downloaded in two ways. As a tarball, ready to be configured and installed on your system (as described in Section 1.1 [Quick start], page 1), see Section 3.2.1 [Release tarball], page 30. If you want official releases of stable versions this is the best, easiest and most common option. Alternatively, you can clone the version controlled history of Gnuastro, run one extra bootstrapping step and then follow the same steps as the tarball. This will give you access to all the most recent work that will be included in the next release along with the full project history. The process is thoroughly introduced in Section 3.2.2 [Version controlled source], page 31.

3.2.1 Release tarball

A release tarball (commonly compressed) is the most common way of obtaining free and open source software. A tarball is a snapshot of one particular moment in the Gnuastro development history along with all the necessary files to configure, build, and install Gnuastro easily (see Section 1.1 [Quick start], page 1). It is very straightforward and needs the least set of dependencies (see Section 3.1.1 [Mandatory dependencies], page 24). Gnuastro has tarballs for official stable releases and pre-releases for testing. See Section 1.5 [Version numbering], page 5, for more on the two types of releases and the formats of the version numbers. The URLs for each type of release are given below.

Official stable releases (`http://ftp.gnu.org/gnu/gnuastro`):
: This URL hosts the official stable releases of Gnuastro. Always use the most recent version (see Section 1.5 [Version numbering], page 5). By clicking on the "Last modified" title of the second column, the files will be sorted by their date which you can also use to find the latest version. It is recommended to use a mirror to download these tarballs, please visit `http://ftpmirror.gnu.org/gnuastro/` and see below.

Pre-release tar-balls (`http://alpha.gnu.org/gnu/gnuastro`):
: This URL contains unofficial pre-release versions of Gnuastro. The pre-release versions of Gnuastro here are for enthusiasts to try out before an official release. If there are problems, or bugs then the testers will inform the developers to fix before the next official release. See Section 1.5 [Version numbering], page 5, to understand how the version numbers here are formatted. If you want to remain even more up-to-date with the developing activities, please clone the version controlled source as described in Section 3.2.2 [Version controlled source], page 31.

Gnuastro's official tarball is released with two formats: Gzip (with suffix `.tar.gz`) and Lzip (with suffix `.tar.lz`). The former is a very well-known and widely used compression program created by GNU and available in most systems. The latter provides a better compression ratio and more robust archival capacity. For example Gnuastro 0.2's tarball was 2.8MB and 4.2MB with Lzip and Gzip respectively. From the Lzip webpage (`http://`

`www.nongnu.org/lzip/lzip.html`): "Lzip is a lossless data compressor with a user interface similar to the one of gzip or bzip2. Lzip can compress about as fast as gzip (`lzip -0`), or compress most files more than bzip2 (`lzip -9`). Decompression speed is intermediate between gzip and bzip2. Lzip is better than gzip and bzip2 from a data recovery perspective." However, Lzip is currently not too common, so it might not be pre-installed in your operating system. Installing it from your operating system's package manager or from source is very easy.

The GNU FTP server is mirrored (has backups) in various locations on the globe (`http://www.gnu.org/order/ftp.html`). You can use the closest mirror to your location for a more faster download. Note that only some mirrors keep track of the pre-release (alpha) tarballs. Also note that if you want to download immediately after and announcement (see Section 1.9 [Announcements], page 11), the mirrors might need some time to synchronize with the main GNU FTP server.

3.2.2 Version controlled source

The publicly distributed Gnuastro tar-ball (for example `gnuastro-X.X.tar.gz`) does not contain the revision history, it is only a snapshot of the source code at one significant instant of Gnuastro's history (specified by the version number, see Section 1.5 [Version numbering], page 5), ready to be configured and built. To be able to develop successfully, the revision history of the code can be very useful to track when something was added or changed, also some updates that are not yet officially released might be in it.

We use Git for the version control of Gnuastro. For those who are not familiar with it, we recommend the Pro Git[9] book. The whole book is publicly available for online reading and downloading and does a wonderful job at explaining the concepts and best practices.

Let's assume you want to keep Gnuastro in the `TOPGNUASTRO` directory (can be any directory, change the value below). The full version controlled history of Gnuastro can be cloned in `TOPGNUASTRO/gnuastro` by running the following commands[10]:

```
$ TOPGNUASTRO=/home/yourname/Research/projects/
$ cd $TOPGNUASTRO
$ git clone git://git.sv.gnu.org/gnuastro.git
```

The `$TOPGNUASTRO/gnuastro` directory will contain hand-written (version controlled) source code for Gnuastro's programs, libraries, this book and the tests. All are divided into sub-directories with standard and very descriptive names. The version controlled files in the top cloned directory are either mainly in capital letters (for example `THANKS` and `README`) or mainly written in small-caps (for example `configure.ac` and `Makefile.am`). The former are non-programming, standard writing for human readers containing high-level information about the whole package. The latter are instructions to customize the GNU build system for Gnuastro.

The cloned Gnuastro source cannot immediately be configured, compiled, or installed since it only contains hand-written files, not automatically generated or imported files which do all the hard work of the build process. See Section 3.2.2.1 [Bootstrapping], page 32,

[9] `https://progit.org/`

[10] If your internet connection is active, but Git complains about the network, it might be due to your network setup not recognizing the Git protocol. In that case use the following URL which uses the HTTP protocol instead: `http://git.sv.gnu.org/r/gnuastro.git`

for the process of generating and importing those files (it is very easy!). Once you have bootstrapped Gnuastro, you can run the standard procedures (in Section 1.1 [Quick start], page 1). Very soon after you have cloned it, Gnuastro's main `master` branch will be updated on the main repository (since the developers are actively working on Gnuastro), for the best practices in keeping your local history in sync with the main repository see Section 3.2.2.2 [Synchronizing], page 33.

3.2.2.1 Bootstrapping

The version controlled source code lacks the source files that we have not written or are automatically built. These automatically generated files are included in the distributed tar ball for each distribution (for example `gnuastro-X.X.tar.gz`, see Section 1.5 [Version numbering], page 5) and make it easy to immediately configure, build, and install Gnuastro. However from the perspective of version control, they are just bloatware and sources of confusion (since they are not changed by Gnuastro developers).

The process of automatically building and importing necessary files into the cloned directory is known as *bootstrapping*. All the instructions for an automatic bootstrapping are available in `bootstrap` and configured using `bootstrap.conf`. `bootstrap` is the only file not written by Gnuastro developers but is under version control to enable simple bootstrapping immediately after cloning. It is maintained by the GNU Portability Library (Gnulib) and this file is an identical copy, so do not make any changes in this file since it will be replaced when Gnulib releases an update. Make all your changes in `bootstrap.conf`.

The bootstrapping process has its own separate set of dependencies, the full list is given in Section 3.1.3 [Bootstrapping dependencies], page 27. They are generally very low-level and used by a very large set of commonly used programs, so they are probably already installed on your system. The simplest way to bootstrap Gnuastro is to simply run the bootstrap script within your cloned Gnuastro directory as shown below. However, please read the next paragraph before doing so (see Section 3.2.2 [Version controlled source], page 31, for `TOPGNUASTRO`).

```
$ cd TOPGNUASTRO/gnuastro
$ ./bootstrap                       # Requires internet connection
```

Without any options, `bootstrap` will clone Gnulib within your cloned Gnuastro directory (`TOPGNUASTRO/gnuastro/gnulib`) and download the necessary Autoconf archives macros. So if you run bootstrap like this, you will need an internet connection every time you decide to bootstrap. Also, Gnulib is a large package and cloning it can be slow. It will also keep the full Gnulib repository within your Gnuastro repository, so if another one of your projects also needs Gnulib, and you insist on running bootstrap like this, you will have two copies. In case you regularly backup your important files, Gnulib will also slow down the backup process. Therefore while the simple invocation above can be used with no problem, it is not recommended, see the next paragraph.

The recommended way to get these two packages is thoroughly discussed in Section 3.1.3 [Bootstrapping dependencies], page 27, (in short: clone them in the separate `DEVDIR/` directory). The following commands will take you into the cloned Gnuastro directory and run the `bootstrap` script, while telling it to copy some files (instead of making symbolic

links, with the `--copy` option, this is not mandatory[11]) and where to look for Gnulib (with the `--gnulib-srcdir` option).

```
$ cd $TOPGNUASTRO/gnuastro
$ ./bootstrap --copy --gnulib-srcdir=$DEVDIR/gnulib
```

Since Gnulib and Autoconf archives are now available in your local directories, you don't need an internet connection every time you decide to remove all untracked files and redo the bootstrap (see box below). You can also use the same command on any other project that uses Gnulib. All the necessary GNU C library functions, Autoconf macros and Automake inputs are now available along with the book figures. The standard GNU build system (Section 1.1 [Quick start], page 1) will do the rest of the job.

Undoing the bootstrap: During the development, it might happen that you want to remove all the automatically generated and imported files. In other words, you might want to reverse the bootstrap process. Fortunately Git has a good program for this job: `git clean`. Run the following command and every file that is not version controlled will be removed.

```
git clean -fxd
```

It is best to commit any recent change before running this command. You might have created new files since the last commit and if they haven't been committed, they will all be gone forever (using `rm`). To get a list of the non-version controlled files instead of deleting them, add the `n` option to `git clean`, so it becomes `-fxdn`.

Besides the `bootstrap` and `bootstrap.conf`, the `bootstrapped/` directory and `README-hacking` file are also related to the bootstrapping process. The former hosts all the imported (bootstrapped) directories. Thus, in the version controlled source, it only contains a `REAME` file, but in the distributed tar-ball it also contains sub-directories filled with all bootstrapped files. `README-hacking` contains a summary of the bootstrapping process discussed in this section. It is a necessary reference when you haven't built this book yet. It is thus not distributed in the Gnuastro tarball.

3.2.2.2 Synchronizing

The bootstrapping script (see Section 3.2.2.1 [Bootstrapping], page 32) is not regularly needed: you mainly need it after you have cloned Gnuastro (once) and whenever you want to re-import the files from Gnulib, or Autoconf archives[12] (not too common). However, Gnuastro developers are constantly working on Gnuastro and are pushing their changes to the official repository. Therefore, your local Gnuastro clone will soon be out-dated. Gnuastro has two mailing lists dedicated to its developing activities (see Section 11.4 [Developing mailing lists], page 240). Subscribing to them can help you decide when to synchronize with the official repository.

To pull all the most recent work in Gnuastro, run the following command from the top Gnuastro directory:

```
$ git pull && autoconf -f
```

[11] The `--copy` option is recommended because some backup systems might do strange things with symbolic links.

[12] `https://savannah.gnu.org/task/index.php?13993` is defined for you to check if significant (for Gnuastro) updates are made in these repositories, since the last time you pulled from them.

GNU Autoconf is part of the GNU build system and will update the `./configure` script based on the hand-written configurations (in `configure.ac`, which is version controlled in Gnuastro). The pulled changes might contain changes in the build system configurations. However, The most important reason for running this command is to generate a version number for your Gnuastro snapshot. This generated version number will include the commit information if you are building Gnuastro from any point in Gnuastro's history (see Section 1.5 [Version numbering], page 5). Since the version number is included in nearly all outputs of the programs, this can help you later exactly reproduce an old result by checking out the exact point in Gnuastro's history that produced those results. Therefore, be sure to run '`autoconf -f`' after every synchronization. You can also run them separately:

```
$ git pull
$ autoconf -f
```

If you would like to see what has changed since you last synchronized your local clone, you can take the following steps instead of the simple command above (don't type anything after `#`):

```
$ git checkout master             # Confirm if you are on master.
$ git fetch origin                # Fetch all new commits from server.
$ git log master..origin/master   # See all the new commit messages.
$ git merge origin/master         # Update your master branch.
$ autoconf -f                     # Update ./configure.
```

By default `git log` prints the most recent commit first, add the `--reverse` option to see the changes chronologically. To see exactly what has been changed in the source code along with the commit message, add a `-p` option to the `git log`.

If you intend make changes in the code, have a look at Chapter 11 [Developing], page 236, to get started easily. Be sure to commit your changes in a separate branch (keep your `master` branch to follow the official repository) and re-run `autoconf -f` after the commit. If you intend to send your changes to us (see Section 11.11 [Contributing to Gnuastro], page 250) for the benefit of the whole community. If you send your work to us, you can safely use your commit since it will be ultimately recorded in Gnuastro's official history. If not, please upload your separate branch to a public hosting service (for example GitLab, see Section 11.11.4 [Forking tutorial], page 254) and link to it in your report, or run `make distcheck` and upload the output `gnuastro-X.X.X.XXXX.tar.gz` to a publicly accessible webpage so your results can be considered scientific (reproducible).

3.3 Build and install

This section is basically a longer explanation to the sequence of commands given in Section 1.1 [Quick start], page 1. If you didn't have any problems during the Section 1.1 [Quick start], page 1, steps, you want to have all the programs of Gnuastro installed in your system, you don't want to change the executable names during or after installation, you have root access to install the programs in the default system wide directory, the Letter paper size of the print book is fine for you or as a summary you don't feel like going into the details when everything is working, you can safely skip this section.

If you have any of the above problems or you want to understand the details for a better control over your build and install, read along. The dependencies which you will need prior to configuring, building and installing Gnuastro are explained in Section 3.1 [Dependencies],

page 24. The first three steps in Section 1.1 [Quick start], page 1, need no extra explanation, so we will skip them and start with an explanation of Gnuastro specific configuration options and a discussion on the installation directory in Section 3.3.1 [Configuring], page 35, followed by some smaller subsections: Section 3.3.2 [Tests], page 42, Section 3.3.3 [A4 print book], page 42, and Section 3.3.4 [Known issues], page 43, which explains the solutions to known problems you might encounter in the installation steps and ways you can solve them.

3.3.1 Configuring

The `$ ./configure` step is the most important step in the build and install process. All the required packages, libraries, headers and environment variables are checked in this step. The behaviors of make and make install can also be set through command line options to this command.

The configure script accepts various arguments and options which enable the final user to highly customize whatever she is building. The options to configure are generally very similar to normal program options explained in Section 4.1.1 [Arguments and options], page 45. Similar to all GNU programs, you can get a full list of the options along with a short explanation by running

```
$ ./configure --help
```

A complete explanation is also included in the `INSTALL` file. Note that this file was written by the authors of GNU Autoconf (which builds the `configure` script), therefore it is common for all programs which use the `$ ./configure` script for building and installing, not just Gnuastro. Here we only discuss cases where you don't have super-user access to the system and if you want to change the executable names. But before that, a review of the options to configure that are particular to Gnuastro are discussed.

3.3.1.1 Gnuastro configure options

Most of the options to configure (which are to do with building) are similar for every program which uses this script. Here the options that are particular to Gnuastro are discussed. The next topics explain the usage of other configure options which can be applied to any program using the GNU build system (through the configure script).

`--enable-progname`
: Only build and install `progname` along with any other program that is enabled in this fashion. `progname` is the name of the executable without the `ast`, for example `imgcrop` for ImageCrop (with the executable name of `astimgcrop`). If this option is called for any of the programs in Gnuastro, any program which is not explicitly enabled will not be built or installed.

`--disable-progname`
`--enable-progname=no`
: Do not build or install the program named `progname`. This is very similar to the `--enable-progname`, but will build and install all the other programs except this one.

`--enable-gnulibcheck`
: Enable checks on the GNU Portability Library (Gnulib). Gnulib is used by Gnuastro to enable users of non-GNU based operating systems (that don't use GNU C library or glibc) to compile and use the advanced features that this

library provides. We make extensive use of such functions. If you give this option to `$ ./configure`, when you run `$ make check`, first the functions in Gnulib will be tested, then the Gnuastro executables. If your operating system does not support glibc or has an older version of it and you have problems in the build process (`$ make`), you can give this flag to configure to see if the problem is caused by Gnulib not supporting your operating system or Gnuastro, see Section 3.3.4 [Known issues], page 43.

`--disable-guide-message`
`--enable-guide-message=no`

Do not print a guiding message during the GNU Build process of Section 1.1 [Quick start], page 1. By default, after each step, a message is printed guiding the user what the next command should be. Therefore, after `./configure`, it will suggest running `make`. After `make`, it will suggest running `make check` and so on. If Gnuastro is configured with this option, for example

```
$ ./configure --disable-guide-message
```

Then these messages will not be printed after any step (like most programs). For people who are not yet fully accustomed to this build system, these guidelines can be very useful and encouraging. However, if you find those messages annoying, use this option.

Note: If some programs are enabled and some are disabled, it is equivalent to simply enabling those that were enabled. Listing the disabled programs is redundant.

Note that the tests of some programs might require other programs to have been installed and tested. For example MakeProfiles is the first program to be tested when you run `$ make check`, it provides the inputs to all the other tests. So if you don't install MakeProfiles, then the tests for all the other programs will be skipped or fail. To avoid this, in one run, you can install all the packages and run the tests but not install. If everything is working correctly, you can run configure again with only the packages you want but not run the tests and directly install after building.

3.3.1.2 Installation directory

One of the most commonly used options to configure is the directory that will host all the files which require installing (for example the actual executable files for the program and/or the documentation and configuration files). This is done through the `--prefix` option. To demonstrate its applicability, let's assume you don't have administrator or root access but you want to be able to access the installed files from anywhere while you are logged in (one of the most common uses of `--prefix`).

The most basic way to run an executable is to explicitly type the full file name (including all the directory information) and run it. One example is running the configuration script with the `$ ./configure` command (see Section 1.1 [Quick start], page 1). By giving a specific directory (the current directory or `./`), we are explicitly telling the shell to look in the current directory for an executable file named '`configure`'. Directly specifying the directory is thus useful for simple shell scripts in the current (or nearby) directories. However, when the program (an executable) is to be used a lot, specifying all those directories will become

a significant burden. For example, the `ls` executable lists the contents in a given directory and it is (usually) installed in the `/usr/bin/` directory by the operating system maintainers. So each time you you want to use it you would have to run the following command (which is very inconvenient, both in typing and in remembering the various directories).

```
$ /usr/bin/ls
```

To address this problem, we have the `PATH` environment variable. To understand it better, we will start with a short introduction to the shell variables. Shell variable values are basically treated as strings of characters. You can define a variable and a value for it by running

```
$ myvariable=a_test_value
```

You can see the value it represents by running

```
$ echo $myvariable
```

If a variable has no value, the last command will only print an empty line. A variable defined like this will be known as long as this shell or terminal is running. Other terminals will have no idea it existed. The main advantage of shell variables is that if they are exported[13], subsequent programs that are run within that shell can access their value. So by giving them different values, you can change the "environment" of a program which uses their values. The shell variables which are accessed by programs are therefore known as "environment variables"[14]. You can see the full list of the environment variables that your shell currently recognizes by running:

```
$ printenv
```

`HOME` is one commonly used environment variable, it is any user's (the one that is logged in) top directory. It is used so often that the shell has a special expansion (alternative) for it: '`~`'. Whenever you see file names starting with the tilde sign, it actually represents the value to the `HOME` environment variable, so `~/doc` is the same as `$HOME/doc`.

Another one of the most commonly used environment variables is `PATH`, it is a list of directories to search for executable names. Its value is a list of directories (separated by a colon, or '`:`'). When the address of the executable is not explicitly given (like `./configure` above), the system will look for the executable in the directories specified by `PATH`. If you have a computer nearby, try running the following command to see which directories your system will look into when it is searching for executable (binary) files, one example is printed here (notice how `/usr/bin`, in the `ls` example above, is one of the directories in `PATH`):

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/bin
```

By default `PATH` usually contains system-wide directories, which are readable (but not writable) by all users, like the above example. Therefore if you don't have root (or administrator) access, you need to add another directory to `PATH` which you actually have write access to. The standard directory where you can keep installed files (not just executables) for your own user is the `~/.local/` directory. The names of hidden files start with a '`.`' (dot), so it will not show up in your common command-line listings, or on the graphical user interface. You can use any other directory, but this is the most recognized.

[13] By running `$ export myvariable=a_test_value` instead of the simpler case in the text

[14] You can use shell variables for other actions too, for example to temporarily keep some names or run loops on some files.

The top installation directory will be used to keep all the package's components: programs (executables), libraries, include (header) files, shared data (like manuals), or configuration files. So it commonly has some of the following sub-directories for each class of components respectively: `bin/`, `lib/`, `include/ man/`, `share/`, `etc/`. Since the `PATH` variable is only used for executables, you can add the `~/.local/bin` directory to `PATH` with the following command. As defined below, first the existing value of `PATH` is used, then your given directory is added to its end and the combined value is put back in `PATH` (run '`$ echo $PATH`' afterwards to check if it was added).

```
$ PATH=$PATH:~/.local/bin
```

Any executable that you installed in this directory will now be usable without having to remember and type its full address. However, as soon as you leave your current terminal session, this modification will be lost. Adding your specified directory to the `PATH` environment variable each time you start a terminal is also very inconvenient and prone to errors. So there are standard 'startup files' defined by your shell. There is a special startup file for every significant starting step:

`/etc/profile` and everything in `/etc/profile.d/`
: These startup scripts are called when your whole system starts (for example after you turn on your computer). Therefore you need administrator or root privileges to access or modify them.

`~/.bash_profile`
: If you are using (GNU) Bash as your shell, the commands in this file are run once every time you log in to your account.

`~/.bashrc`
: If you are using (GNU) Bash as your shell, the commands here will be run each time you start a terminal (for example, when you open your terminal emulator in the graphic user interface).

For security reasons, it is highly recommended to directly type in your `HOME` directory value by hand in startup files instead of using variables. So in the following, let's assume your user name is '`name`'. To add `~/.local/bin` to your `PATH` automatically on every startup you have to "export" the new value of `PATH` in the startup file that is most relevant to you with the line '`export PATH=$PATH:/home/name/.local/bin`'. You can either do it manually using a text editor, or by running the following command which will add this line as the last line of the file. Let's assume you want to add it to `~/.bashrc` (afterwards, open your `~/.bashrc` with a text editor and check it out to see for your self):

```
$ echo 'export PATH=$PATH:/home/name/.local/bin' >> ~/.bashrc
```

Now that you know your system will look into `~/.local/bin` for executables, you can tell Gnuastro's configure script to install everything in the top `~/.local` directory using the `--prefix` option. The configure script will then put the executables in `~/.local/bin`, the compiled library files in `~/.local/lib`, the library header files in `~/.local/include` and so on (see Section 10.1 [Review of library fundamentals], page 189, for a review of the compiled library files and the headers). When you subsequently run `$ make install`, all the install-able files will be put in their respective directory under `~/.local/` (as discussed

above). Note that tilde (‘~’) expansion will not happen if you put a ‘=’ between `--prefix` and `~/.local`[15], so we have avoided it here, see Section 4.1.3 [Options], page 46.

```
$ ./configure --prefix ~/.local
```

You can install everything (including libraries like GSL, CFITSIO, or WCSLIB) locally by configuring them as above. However, recall that `PATH` is only for executable files, not libraries. Therefore, when building programs or libraries[16] that depend on libraries you installed like this, you have to guide your compiler to the necessary directories. To do that, you have to define the `LDFLAGS` and `CPPFLAGS` environment variables respectively. This can be done while calling `./configure`:

```
$ ./configure LDFLAGS=-L/home/name/.local/lib            \
              CPPFLAGS=-I/home/name/.local/include       \
              --prefix ~/.local
```

It can be annoying to do this when configuring every package, so you can define these two variables in a startup file (discussed above). The convention on using these variables doesn't include a colon to separate values (as `PATH`-like variables do), they use white space characters and each value is prefixed with a compiler option[17] (note the `-L` and `-I` above, see Section 4.1.3 [Options], page 46). Therefore we have to keep the value in double quotation signs to keep the white space characters. Run the following two commands to do that if you want them in your `~/.bashrc`.

```
echo 'export LDFLAGS="$LDFLAGS -L/home/name/.local/lib"' >> ~/.bashrc
echo 'export CPPFLAGS="$CPPFLAGS -I/home/name/.local/include"' \
     >> ~/.bashrc
```

To be able to link to the locally installed dynamic libraries (which are linked at run time) after installation, add `~/.local/lib` to your `LD_LIBRARY_PATH` environment variable similar to the example above for `PATH`. If you also want to access the Info (see Section 4.6.4 [Info], page 60) and man pages (see Section 4.6.3 [Man pages], page 59) documentations add `~/.local/share/info` and `~/.local/share/man` to your `INFOPATH`[18] and `MANPATH` environment variables.

A final note is that order matters in the directories that are searched for all the variables discussed above. In the examples above, the new directory was added after the system specified directories. So if the program, library or manuals are found in the system wide directories, the user directory is no longer searched. If you want to search your local installation first, put the new directory before the already existing list like the example below.

```
export LD_LIBRARY_PATH=/home/name/.local/lib:$LD_LIBRARY_PATH
```

[15] If you insist on using ‘=’, you can use `--prefix=$HOME/.local`.

[16] For example WCSLIB which needs CFITSIO, or Gnuastro which needs both.

[17] These variables are ultimately used as options while building the programs, so every value has be an option name followed be a value as discussed in Section 4.1.3 [Options], page 46.

[18] Info has the following convention: “If the value of `INFOPATH` ends with a colon [or it isn't defined] ..., the initial list of directories is constructed by appending the build-time default to the value of `INFOPATH`.” So when installing in a non-standard directory and if `INFOPATH` was not initially defined, add a colon to the end of `INFOPATH` as shown below, otherwise Info will not be able to find system-wide installed documentation:
`echo 'export INFOPATH=$INFOPATH:/home/name/.local/share/info:' >> ~/.bashrc`
Note that this is only an internal convention of Info, do not use it for other `*PATH` variables.

This is good when a library, for example CFITSIO, is already present on the system, but the system-wide install wasn't configured with the correct configuration flags (see Section 3.1.1.2 [CFITSIO], page 25), or you want to use a newer version and you don't have administrator or root access to update it. With e above order `LD_LIBRARY_PATH`, the system will first find the CFITSIO you installed for yourself and will never reach the system-wide installation. However there are important security problems: because all imporatant system-wide programs and libraries can be replaced by non-secure versions if they also exist in `./.local/`. So if you choose this order, be sure to keep it clean from executables or libraries with the same names as important system programs or libraries.

3.3.1.3 Executable names

At first sight, the names of the executables for each program might seem to be uncommonly long, for example `astnoisechisel` or `astimgcrop`. We could have chosen terse (and cryptic) names like most programs do. We chose this complete naming convention (something like the commands in TEX) so you don't have to spend too much time remembering what the name of a specific program was. Such complete names also enable you to easily search for the programs.

To facilitate typing the names in, we suggest using the shell auto-complete. With this facility you can find the executable you want very easily. It is very similar to file name completion in the shell. For example, simply by typing the letters below (where `[TAB]` stands for the Tab key on your keyboard)

```
$ ast[TAB][TAB]
```

you will get the list of all the available executables that start with `ast` in your `PATH` environment variable directories. So, all the Gnuastro executables installed on your system will be listed. Typing the next letter for the specific program you want along with a Tab, will limit this list until you get to your desired program.

In case all of this does not convince you and you still want to type short names, some suggestions are given below. You should have in mind though, that if you are writing a shell script that you might want to pass on to others, it is best to use the standard name because other users might not have adopted the same customizations. The long names also serve as a form of documentation in such scripts. A similar reasoning can be given for option names in scripts: it is good practice to always use the long formats of the options in shell scripts, see Section 4.1.3 [Options], page 46.

The simplest solution is making a symbolic link to the actual executable. For example let's assume you want to type `ic` to run ImageCrop instead of `astimgcrop`. Assuming you installed Gnuastro executables in `/usr/local/bin` (default) you can do this simply by running the following command as root:

```
# ln -s /usr/local/bin/astimgcrop /usr/local/bin/ic
```

In case you update Gnuastro and a new version of ImageCrop is installed, the default executable name is the same, so your custom symbolic link still works.

The installed executable names can also be set using options to `$ ./configure`, see Section 3.3.1 [Configuring], page 35. GNU Autoconf (which configures Gnuastro for your particular system), allows the builder to change the name of programs with the three options `--program-prefix`, `--program-suffix` and `--program-transform-name`. The first two are for adding a fixed prefix or suffix to all the programs that will be installed. This will

actually make all the names longer! You can use it to add versions of program names to the programs in order to simultaneously have two executable versions of a program.

The third configure option allows you to set the executable name at install time using the SED program. SED is a very useful 'stream editor'. There are various resources on the internet to use it effectively. However, we should caution that using configure options will change the actual executable name of the installed program and on every re-install (an update for example), you have to also add this option to keep the old executable name updated. Also note that the documentation or configuration files do not change from their standard names either.

For example, let's assume that typing `ast` on every invocation of every program is really annoying you! You can remove this prefix from all the executables at configure time by adding this option:

```
$ ./configure --program-transform-name='s/ast/ /'
```

3.3.1.4 Configure and build in RAM

The configure and build process involves the creation, reading, and modification of a large number of files (input/output, or I/O). Therefore file I/O issues can directly affect the work of developers who need to configure and build Gnuastro numerous times. Some of these issues are listed below:

- I/O will cause wear and tear on both the HDDs (mechanical failures) and SSDs (decreasing the lifetime).
- Having the built files mixed with the source files can greatly affect backing up (synchronization) of source files (since it involves the management of a large number of small files that are regularly changed. Backup software can ofcourse be configured to ignore the built files and directories. However, since the built files are mixed with the source files and can have a large variety, this will require a high level of customization.

One solution to address both these problems is to use the tmpfs file system (`https://en.wikipedia.org/wiki/Tmpfs`). Any file in tmpfs is actually stored in the RAM (and possibly SAWP), not on HDDs or SSDs. The RAM is built for extensive and fast I/O. Therefore the large number of file I/Os associated with configuring and building will not harm the HDDs or SSDs. Due to the volatile nature of RAM, files in the tmpfs filesystem will be permanently lost after a power-off. Since all configured and built files are derivative files (not files that have been directly written by hand) there is no problem in this and this feature can be considered as an automatic cleanup.

The modern GNU C library (and thus the Linux kernel) define the `/dev/shm` directory for this purpose (POSIX shared memory). So using GNU Build System's ability to build in a separate directory (not necessarily in the source directory), we can configure and build the programs in `/dev/shm` to benefit from the RAM. To simplify the process, Gnuastro comes with a `tmpfs-config-make` script. This script will create a directory in the shared memory, and put a symbolic link to it (called `build`) in the top source directory (the backup/sync software therefore only needs to ignore this single link/file). The script will then internally change to that directory and configure and build (`make -kjN`, where `N` is the number of threads for a parallel build) Gnuastro in there. To benefit from this script, simply run the following command instead of `./configure` and `make`:

```
$ ./tmpfs-config-make
```

After this script is finished, you can '`cd build`' and run other Make commands (for example, '`make check`', '`make install`', or '`make pdf`') from there. In Emacs, the command to be run with the `M-x compile` command (by default: `make -k`) can be changed to '`cd build; make -kjN`', or '`make -C build -kjN`' (`N` is the number of threads; an integer ≥ 1). For subsequent builds (during development) the `M-x recompile` command will also do all the building in the RAM while you modify the clean, and backed-up source files and make minimal/efficient use of your non-volatile HDD or SSD.

This script can be used in any software which is configured and built using the GNU Build System. Just copy it in the top source directory of that software and run it from there. The default number of threads and location of the shared memory (`/dev/shm`) are currently hard-coded within the script. If you need to change them, please open the script with a text editor and set their values manually.

3.3.2 Tests

After successfully building (compiling) the programs with the `$ make` command you can check the installation before installing. To run the tests, run

```
$ make check
```

For every program some tests are designed to check some possible operations. Running the command above will run those tests and give you a final report. If everything is ok and you have built all the programs, all the tests should pass. In case any of the tests fail, please have a look at Section 3.3.4 [Known issues], page 43, and if that still doesn't fix your problem, look that the `./tests/test-suite.log` file to see if the source of the error is something particular to your system or more general. If you feel it is general, please contact us because it might be a bug. Note that the tests of some programs depend on the outputs of other program's tests, so if you have not installed them they might be skipped or fail. Prior to releasing every distribution all these tests are checked. If you have a reasonably modern terminal, the outputs of the successful tests will be colored green and the failed ones will be colored red.

These scripts can also act as a good set of examples for you to see how the programs are run. All the tests are in the `tests/` directory. The tests for each program are shell scripts (ending with `.sh`) in a sub-directory of this directory with the same name as the program. See Section 11.9 [Test scripts], page 249, for more detailed information about these scripts in case you want to inspect them.

3.3.3 A4 print book

The default print version of this book is provided in the letter paper size. If you would like to have the print version of this book on paper and you are living in a country which uses A4, then you can rebuild the book. The great thing about the GNU build system is that the book source code which is in Texinfo is also distributed with the program source code, enabling you to do such customizations (hacking).

In order to change the paper size, you will need to have GNU Texinfo installed. Open `doc/gnuastro.texi` with any text editor. This is the source file that created this book. In the first few lines you will see this line:

```
@c@afourpaper
```

In Texinfo, a line is commented with `@c`. Therefore, uncomment this line by deleting the first two characters such that it changes to:

```
@afourpaper
```

Save the file and close it. You can now run

```
$ make pdf
```

and the new PDF book will be available in `SRCdir/doc/gnuastro.pdf`. By changing the `pdf` in `$ make pdf` to `ps` or `dvi` you can have the book in those formats. Note that you can do this for any book that is in Texinfo format, they might not have `@afourpaper` line, so you can add it close to the top of the Texinfo source file.

3.3.4 Known issues

Depending on your operating system and the version of the compiler you are using, you might confront some known problems during the configuration (`$ ./configure`), compilation (`$ make`) and tests (`$ make check`). Here, their solutions are discussed.

- `$ ./configure`: *Configure complains about not finding a library even though you have installed it.* The possible solution is based on how you installed the package:
  - From your distribution's package manager. Most probably this is because your distribution has separated the header files of a library from the library parts. Please also install the 'development' packages for those libraries too. Just add a `-dev` or `-devel` to the end of the package name and re-run the package manager. This will not happen if you install the libraries from source. When installed from source, the headers are also installed.
  - From source. Then your linker is not looking where you installed the library. If you followed the instructions in this chapter, all the libraries will be installed in `/usr/local/lib`. So you have to tell your linker to look in this directory. To do so, add `LDFLAGS=-L/usr/local/lib` to the Gnuastro configure script. If you want to use the libraries for your other programming projects, then export this environment variable similar to the case for `LD_LIBRARY_PATH` explained below.
- `$ make`: *Complains about an unknown function on a non-GNU based operating system.* In this case, please run `$ ./configure` with the `--enable-gnulibcheck` option to see if the problem is from the GNU Portability Library (Gnulib) not supporting your system or if there is a problem in Gnuastro, see Section 3.3.1.1 [Gnuastro configure options], page 35. If the problem is not in Gnulib and after all its tests you get the same complaint from `make`, then please contact us at `bug-gnuastro@gnu.org`. The cause is probably that a function that we have used is not supported by your operating system and we didn't included it along with the source tar ball. If the function is available in Gnulib, it can be fixed immediately.
- `$ make`: *Can't find the headers (.h files) of libraries installed from source.* Similar to the case for `LDFLAGS` (above), your compiler is not looking in the right place, add `CPPFLAGS=-I/usr/local/include` to `./configure` to re-configure Gnuastro, then re-run make.
- `$ make check`: *Only one (the first) test passes, all the rest fail.* It is highly likely that your distribution doesn't look into the `/usr/local/lib` directory when searching for shared libraries. To make sure it is added to the list of directories, run the following

command and restart your terminal: (you can ignore the `\` and extra space if you type it, it is only necessary if you copy and paste). See Section 3.3.1.2 [Installation directory], page 36, for more details.

```
echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib' \
     >> ~/.bashrc
```

- `$ make check`: *The tests relying on external programs (for example* `fitstopdf.sh` *fail.)* This is probably due to the fact that the version number of the external programs is too old for the tests we have preformed. Please update the program to a more recent version. For example to create a PDF image, you will need GPL Ghostscript, but older versions do not work, we have successfully tested it on version 9.15. Older versions might cause a failure in the test result.
- `$ make pdf`: *The PDF book cannot be made.* To make a PDF book, you need to have the GNU Texinfo program (like any program, the more recent the better). A working TEX program is also necessary, which you can get from Tex Live[19].

If your problem was not listed above, please file a bug report (Section 1.7 [Report a bug], page 9).

[19] `https://www.tug.org/texlive/`

4 Common program behavior

There are some facts that are common to all the programs in Gnuastro which are mainly to do with user interaction. In this chapter these aspects are discussed. The most basic are the command-line options which are common in all the programs for a unified user experience. All Gnuastro programs can use configuration files so you don't have to specify all the parameters on the command-line each time you run a program. The manner of setting, checking and using the these files at various levels are also explained. Finally we discuss how you can get immediate and distraction-free (without taking your hands off the keyboard!) help on the command-line.

4.1 Command-line

All the programs in Gnuastro are customized through the standard GNU style command-line options. First a general outline of how to make best use of these options is discussed and finally the options that are common to all the programs in Gnuastro are listed.

Your full command-line text is passed onto the shell as a string of characters. That string is then broken up into separate 'words' by any 'metacharacters' (like space, tab, `|`, `>` or `;`) that might exist in the text. See the GNU Bash manual, for the complete list of meta-characters and other Bash definitions. Its "Shell Operation" section has a short summary of the steps the shell takes before passing the commands to the program you called.

4.1.1 Arguments and options

On the command-line, the first thing you enter is the name of the program you want to run. After that you can specify two types of input: *arguments* and *options*. Arguments are those tokens that are not preceded by any hyphens (`-`), the program is suppose to understand what they are without any help from the user.

Arguments can be both mandatory and optional and since there is no help from you, their order might also matter (for example in `cp` which is used for copying). The outputs of `--usage` and `--help` shows which arguments are optional and which are mandatory, see Section 4.6.1 [`--usage`], page 58. As their name suggests, *options* are only optional and most of the time you don't have to worry about what order you specify them in.

In case your arguments or option values contain any of the shell's meta-characters, you have to quote them. If there is only one such character, you can use a backslash (`\`) before it. If there are multiple, it might be easier to simply put your whole argument or option value inside of double quotes (`"`). In such cases, everything inside the double quotes will be seen as one 'word'.

For example let's say you want to specify the Header data unit (HDU) of your FITS file using a complex expression like `3; images(exposure > 100)`. If you simply add these after the `--hdu` (`-h`) option, the programs in Gnuastro will read the value to the HDU option as `3` and run. Then, Bash will attempt to run a separate command `images(exposure > 100)` and complain about a syntax error. This is because the semicolon (`;`) is an 'end of command' character in Bash. To solve this problem you can simply put double quotes around the whole string you want to pass as seen below:

```
$ astimgcrop --hdu="3; images(exposure > 100)" FITSimage.fits
```

Alternatively you can put a \ before every metacharacter in this string, but probably you will agree with us that the double quotes are much more easier, elegant and readable.

4.1.2 Arguments

In Gnuastro, the names of the input data files and ASCII tables are mostly specified as arguments, you can generally specify them in any order unless otherwise stated for a particular program. Everything particular about how a program treats arguments, is explained under the "Invoking ProgramName" section for that program.

Generally, if there is a standard file name extension for a particular format, that filename extension is used to separate the kinds of arguments. The list below shows what astronomical data formats are recognized based on their file name endings. If the program doesn't accept any other data format, any other argument that doesn't end with the specified extensions below is considered to be a text file (usually catalogs). For example Section 5.2 [ConvertType], page 65, accepts other data formats.

- `.fits`: The standard file name ending of a FITS image.
- `.fits.Z`: A FITS image compressed with `compress`.
- `.fits.gz`: A FITS image compressed with GNU zip (gzip).
- `.fits.fz`: A FITS image compressed with `fpack`.
- `.imh`: IRAF format image file.

Through out this book and in the command-line outputs, whenever we want to generalize all such astronomical data formats in a text place holder, we will use `ASTRdata`, we will assume that the extension is also part of this name. Any file ending with these names is directly passed on to CFITSIO to read. Therefore you don't necessarily have to have these files on your computer, they can also be located on an FTP or HTTP server too, see the CFITSIO manual for more information.

CFITSIO has its own error reporting techniques, if your input file(s) cannot be opened, or read, those errors will be printed prior to the final error by Gnuastro.

4.1.3 Options

Command-line options allow configuring the behavior of a program in all GNU/Linux applications for each particular execution. Most options can be called in two ways: *short* or *long* a small number of options in some programs only have the latter type. In the list of options provided in Section 4.1.4 [Common options], page 48, or those for each program, both formats are shown for those which support both. First the short is shown then the long. Short options are only one character and only have one hyphen (for example `-h`) while long options have two hyphens an can have many characters (for example `--hdu`).

Usually, the short options are for when you are writing on the command line and want to save keystrokes and time. The long options are good for shell scripts, where you don't usually have a rush and they provide a level of documentation, since they are less cryptic. Usually after a few months of not running a program, the short options will be forgotten and reading your previously written script will not be easy.

Some options need to be given a value if they are called and some don't. You can think of the latter type of options as on/off options. These two types of options can be distinguished using the output of the `--help` and `--usage` options, which are common to

all GNU software, see Section 4.6 [Getting help], page 57. The following convention is used for the formats of the values in Gnuastro:

`INT` The value is read as an integer. If a float or a string is provided the program will warn you and abort. In most cases, integers are used for counting variables, so if they are negative the program will also abort.

`4or8` Either the value 4 or 8, any other integer will give a warning and abort.

`FLT` The value is read as a float. There are generally two types, depending on the context. If they are for fractions, they will have to be less than or equal to unity.

`STR` The value is read as a string of characters (for example a file name) or other particular settings like a HDU name, see below.

To specify a value in the short format, simply put the value after the option. Note that since the short options are only one character long, you don't have to type anything between the option and its value. For the long option you either need white space or an `=` sign, for example `-h2`, `-h 2`, `--hdu 2` or `--hdu=2` are all equivalent.

The short format of on/off options (those that don't need values) can be concatenated for example these two hypothetical sequences of options are equivalent: `-a -b -c4` and `-abc4`. As an example, consider the following command to run ImageCrop:

```
$ astimgcrop -Dr3 --wwidth 3 catalog.txt --deccol=4 ASTRdata
```

The `$` is the shell prompt, `astimgcrop` is the program name. There are two arguments (`catalog.txt` and `ASTRdata`) and four options, two of them given in short format (`-D`, `-r`) and two in long format (`--width` and `--deccol`). Three of them require a value and one (`-D`) is an on/off option.

If an abbreviation is unique between all the options of a program, the long option names can be abbreviated. For example, instead of typing `--printparams`, typing `--print` or maybe even `--pri` will be enough, if there are conflicts, the program will warn you and show you the alternatives. Finally, if you want the argument parser to stop parsing arguments beyond a certain point, you can use two dashes: `--`. No text on the command-line beyond these two dashes will be parsed.

If an option with a value is repeated or called more than once, the value of the last time it was called will be assigned to it. This very useful in complicated situations, for example in scripts. Let's say you want to make a small modification to one option value. You can simply type the option with a new value in the end of the command and see how the script works. If you are satisfied with the change, you can remove the original option. If the change wasn't satisfactory, you can remove the one you just added and not worry about saving the original value. Without this capability, you would have to memorize or save the original value somewhere else, run the command and then change the value again which is not at all convenient and is potentially cause lots of bugs.

When you don't call an option that requires a value, all the programs in Gnuastro will check configuration files to find a value for that parameter. To learn more about how folder, user and system wide configuration files can be set, please see Section 4.2 [Configuration files], page 51. Another factor that is particular to Gnuastro is that it will check the value you have given for each option to see if it is reasonable. For example you might mistakenly

give a negative, float or string value for a FITS image extension or column number. As another example, you might give a value larger than unity for an option that only accepts fractions (which are always less than unity and positive).

CAUTION: In specifying a file address, if you want to use the shell's tilde expansion (`~`) to specify your home directory, leave at least one space between the option name and your value. For example use `-o ~/test`, `--output ~/test` or `--output= ~/test`. Calling them with `-o~/test` or `--output=~/test` will disable shell expansion.

CAUTION: If you forget to specify a value for an option which requires one, and that option is the last one, Gnuastro will warn you. But if it is in the middle of the command, it will take the text of the next option or argument as the value which can cause undefined behavior.

NOTE: All counting in Gnuastro starts from 0 not 1. So for example the first FITS image extension or column in a table are noted by 0, not 1. This is the standard in C and all languages that are based on it (for example C++, Java and Python).

4.1.4 Common options

To facilitate the job of the users and developers, all the programs in Gnuastro share some basic command-line options for the same operations where they are relevant. The list of options is provided below. It is noteworthy that these similar options are hard-wired into the programming of all of Gnuastro programs using GNU C library's argument parser merging ability.

For some programs, some of the options, might be irrelevant for example MakeProfiles creates FITS images based on a given catalog. Therefore no input images (and thus HDUs) are necessary for it. In such cases, the option is still listed and if a value is given for it, it is completely ignored.

4.1.4.1 Input/Output options

These options are to do with the input and outputs of the various programs.

`-h`
`--hdu` (`=STR`) The number or name of the desired Header Data Unit or HDU in the input FITS image or images. A FITS file can store multiple HDUs or extensions, each with either an image or a table or nothing at all (only a header). Note that counting of the extensions starts from 0(zero), not 1(one). When specifying the name, case is not important so `IMAGE`, `image` or `ImAgE` are equivalent.

A `#` is appended to the string you specify for the HDU[1] and the result is put in square brackets and appended to the FITS file name before calling CFITSIO to read the contents of the HDU for all the programs in Gnuastro. CFITSIO

[1] With the `#` character, CFITSIO will only read the desired HDU into your memory, not all the existing HDUs in the fits file.

has many capabilities to help you find the extension you want, far beyond the simple extension number and name. See CFITSIO manual's "HDU Location Specification" section for a very complete explanation with several examples.

Note that in some programs, the behavior of `--hdu` might be different, for example see Section 6.2.3 [Invoking Arithmetic], page 86, were it can be called multiple times and the value of each all will be stored.

`-o`
`--output` (`=STR`) The name of the output file or directory. With this option the automatic output names explained in Section 4.5 [Automatic output], page 57, are ignored.

`-D`
`--dontdelete`
By default, if the output file already exists, it will be silently replaced with the output of this run of all Gnuastro programs. By calling this option, if the output file already exists, the programs will warn you and abort.

`-K`
`--keepinputdir`
In automatic output names, don't remove the directory information of the input file names. As explained in Section 4.5 [Automatic output], page 57, if no output name is specified, then the output name will be made in the existing directory based on your input. If you call this option, the directory information of the input will be kept and the output will be in the same directory as the input. Note that his is only relevant if you are running the program from another directory!

4.1.4.2 Operating modes

Another group of options that are common to all the programs in Gnuastro are those to do with the general operation of the programs. The explanation for those that are not only limited to Gnuastro but can be called in all GNU programs start with (GNU option).

`--` (GNU option) Stop parsing the command-line. This option can be useful in scripts or when using the shell history. Suppose you have a long list of options, and want to see if removing some of them (and using the default values) can give a better result. If the ones you want to remove are the last ones on the command-line, you don't have to delete them, you can just add `--` before them and if you don't get what you want, you can remove the `--` and get the same initial result.

`--usage` (GNU option) Only print the options and arguments. This is very useful for when you know the what the options do, you have just forgot their names. See Section 4.6.1 [`--usage`], page 58.

`-?`
`--help` (GNU option) Print all options and an explanation. Adding this option will print all the options in their short and long formats, also displaying which ones need a value if they are called (with an `=` after the long format). A short explanation is also given for what the option is for. The program will quit

immediately after the message is printed and will not do any form of processing. See Section 4.6.2 [--help], page 58.

`-V`
`--version`
(GNU option) Print a short message, showing the full name, version, copyright information and program authors. On the first line it will print the official name (not executable name) and version number of the program. It will also print the version of the Gnuastro that the program was built with. Following this is a blank line and a copyright information. The program will not run.

`-q`
`--quiet` Don't report steps. All the programs in Gnuastro that have multiple major steps will report their steps for you to follow while they are operating. If you do not want to see these reports, you can call this option and only error messages will be printed if the program is aborted. If the steps are done very fast (depending on the properties of your input) disabling these reports will also decrease running time.

`--cite` Print the BibTeX entry for Gnuastro and the particular program (if that program comes with a separate paper) and abort. Citations are vital for the continued work on Gnuastro. Gnuastro started and is continued based on separate research projects. So if you find any of the tools offered in Gnuastro to be useful in your research, please use the output of this command to cite the program and Gnuastro in your research paper. Thank you.

Gnuastro is still new, there is no separate paper only devoted to Gnuastro yet. Therefore currently the paper to cite for Gnuastro is the paper for NoiseChisel which is the first published paper introducing Gnuastro to the astronomical community. Upon reaching a certain point, a paper completely devoted to Gnuastro will be published, see Section 1.5.1 [GNU Astronomy Utilities 1.0], page 6.

`-P`
`--printparams`
Print the final values used for all the parameters and abort. See Section 4.4 [Final parameter values, reproduce previous results], page 56, for more details.

`-S`
`--setdirconf`
Update the current directory configuration file from the given command line options and quit, see Section 4.2 [Configuration files], page 51. The values of your options are added to the configuration file in the current directory. If the configuration file or folder doesn't exist, it will be created. If it exists but has different values for those options, they will be given the new values. In any case, the program will not run, but the contents of its updated configuration file are printed for you to inspect.

This is the recommended method to fill the configuration file for all future calls to one of the Gnuastro programs in a folder. It will internally check if your values are in the correct range and type and save them according to the configuration file format, see Section 4.2.1 [Configuration file format], page 52.

When this option is called, the otherwise mandatory arguments, for example input image or catalog file(s), are no longer mandatory (since the program will not run).

`-U`
`--setusrconf`
Update the user configuration file from the command-line options and quit. See explanation under `--setdirconf` for more details.

`--onlydirconf`
Only read the current (local) directory configuration file and ignore the rest of the configuration files, see Section 4.2.2 [Configuration file precedence], page 52, and Section 4.2.3 [Current directory and User wide], page 53. This can be very useful when you want your results to be exactly reproducible. All the configuration files can be put in the hidden `./.gnuastro/` directory in the current directory, or the hidden directory can be a symbolic link to the directory containing the configuration files. Then with this option you can ensure that no other configuration file is read. So if your local configuration file lacks some parameters, which ever Gnuastro program you are using will will warn you and abort, enabling you to exactly set all the necessary parameters without unknowingly relying on some user or system wide option values.

`onlydirconf` can also be used in the configuration files (with a value of `0` or `1`), see Section 4.2.1 [Configuration file format], page 52. If it is present in the local configuration file, other configuration files will not be read. In the other configuration files, it is irrelevant.

`--onlyversion`
(`=STR`) Only run the program if its version is equal with the string of characters given to this option. Note that it is not compared as a number, but as a string of characters, so `0`, or `0.0` and `0.00` are different. This is useful if you want your results to be exactly reproducible and not mistakenly run with an updated or older version of the program.

`--nolog` For programs which generate Log files, if this option is called, no Log file will be generated.

`-N`
`--numthreads`
(`=INT`) Set the number of CPU threads to use. See Section 4.3 [Threads in Gnuastro], page 54.

4.2 Configuration files

Each program needs a certain number of parameters to run. Supplying all the necessary parameters each time you run the program is very frustrating and prone to errors. Therefore all the programs read the values for the necessary options you have not given in the command line from one of several plain text files (which you can view and edit with any text editor). These files are known as configuration files and are usually kept in a directory named `etc/` according to the file system hierarchy standard[2].

[2] `http://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard`

The thing to have in mind is that none of the programs in Gnuastro keep any internal default value. All the values must either be stored in one of the configuration files or explicitly called in the command-line. In case the necessary parameters are not given through any of these methods, the program will list the missing necessary parameters and abort. The only exception to this is `--numthreads`, whose default value is determined at run-time using the number of threads available to your system, see Section 4.3 [Threads in Gnuastro], page 54. Of course, you can still provide a default value for the number of threads at any of the levels below, but if you don't, the program will not abort. Also note that through automatic output name generation, the value to the `--output` option is also not mandatory on the command-line or in the configuration files for all programs which don't rely on that value as an input[3], see Section 4.5 [Automatic output], page 57.

4.2.1 Configuration file format

The configuration files for each program have the standard program executable name with a `.conf` suffix. When you download the source code, you can find them in the same directory as the source code of each program, see Section 11.6 [Program source], page 244.

Any line in the configuration file whose first non-white character is a `#` is considered to be a comment and is ignored. The same goes for an empty line. The name of the parameter is the same as the long format of the command-line option for that parameter. The parameter name and parameter value have to be separated by any number of 'white-space' characters: space, tab or vertical tab. By default several space characters are used. If the value of an option has space characters (most commonly for the `hdu` option), then double quotes can be used to specify the full value.

In each non-commented and non-blank line, any text after the first two words (given the conditions above) is ignored. If it is an option without a value in the `--help` output (on/off option, see Section 4.1.3 [Options], page 46), then the value should be `1` if it is to be 'on' and `0` otherwise. If an option is not recognized in the configuration file, the name of the file and unrecognized option will be reported and the program will abort. If a parameter is repeated more more than once in the configuration files and it is not set on the command-line, then only the first value will be used, the rest will be ignored.

You can build or edit any of the directories and the configuration files yourself using any text editor. However, it is recommended to use the `--setdirconf` and `--setusrconf` options to set default values for the current directory or this user, see Section 4.1.4.2 [Operating modes], page 49. With these options, the values you give will be checked as explained in Section 4.1.3 [Options], page 46, before writing in the current directory's configuration file. They will also print a set of commented lines guiding the reader and will also classify the options based on their context and write them in their logical order to be more understandable.

4.2.2 Configuration file precedence

The parameter values in all the programs of Gnuastro will be filled in the following order. Such that if a parameter is assigned a value in an earlier step, any value for that parameter in a later step will be ignored.

[3] One example of a program which uses the value given to `--output` as an input is ConvertType, this value specifies the type of the output through the value to `--output`, see Section 5.2.3 [Invoking ConvertType], page 68.

1. Command-line options, for this particular execution.
2. Current directory, for all executions in the directory from which any of the programs is run (`./.gnuastro/`).
3. The user's home directory, for all the executions of a particular user: (`$HOME/.local/etc/`, see below). It is only read if `--onlydirconf` is not called, see Section 4.1.4.2 [Operating modes], page 49.
4. In a system wide directory for any user on that computer (`prefix/etc/`, see Section 3.3.1.2 [Installation directory], page 36, for the value of `prefix`). It is only read if `--onlydirconf` is not called, see Section 4.1.4.2 [Operating modes], page 49.

The basic idea behind setting this progressive state of checking for parameter values is that separate users of a computer or separate folders in a user's file system might need different values for some parameters and the same values for others. For example raw telescope images usually have their main image extension in the second FITS extension, while processed FITS images usually only have one extension. If your system wide default input extension is 0 (the first), then when you want to work with the former group of data you have to explicitly mention it to the programs every time. With this progressive state of default values to check, you can set different default values for the different directories that you would like to run Gnuastro in for your different purposes, so you won't have to worry about this issue any more.

4.2.3 Current directory and User wide

For the current (local) and user-wide directories, the configuration files are stored in the hidden sub-directories named `./.gnuastro/` and `HOME/.local/etc/` respectively. Unless you have changed it, the `HOME` environment variable should point to your home directory. You can check it by running `$ echo $HOME`. Each time you run any of the programs in Gnuastro, this environment variable is read and placed in the above address. So if you suddenly see that your home configuration files are not being read, probably you (or some other program) has changed the value of this environment variable.

Although it might cause confusions like above, this dependence on the `HOME` environment variable enables you to temporarily use a different directory as your home directory. This can come in handy in complicated situations. To set the user or current directory configuration files based on your command-line input, you can use the `--setdirconf` or `--setusrconf`, see Section 4.1.4.2 [Operating modes], page 49,

4.2.4 System wide

When Gnuastro is installed, the configuration files that are shipped with the distribution are copied into the (possibly system wide) `prefix/etc/` directory. See Section 3.3.1 [Configuring], page 35, for more details on `prefix` (by default it is: `/usr/local`). This directory is the final place (with the lowest priority) that the programs in Gnuastro will check to retrieve parameter values.

If you remove a parameter and its value from the files in this system wide directory, you either have to specify it in more immediate configuration files or set it each time in the command-line. Recall that none of the programs in Gnuastro keep any internal default values and will abort if they don't find a value for the necessary parameters (except the

number of threads). So even though you might never use a parameter, it still has to be at least available in this system-wide configuration file.

In case you install Gnuastro from your distribution's repositories, `prefix` will either be set to `/` (the root directory) or `/usr`, so you can find the system wide configuration variables in `/etc/` or `/usr/etc/`. The prefix of `/usr/local/` is conventionally used for programs you install from source by your self.

4.3 Threads in Gnuastro

Some of the programs benefit significantly when you use all the threads your computer's CPU has to offer to your operating system. GNU Astronomy Utilities uses the POSIX threads library (pthreads) for spinning off threads when the user asks for it. The number of threads available to your operating system is usually double the number of physical (hardware) cores in your CPU. On a GNU/Linux system, the number of threads available can be found with the command `$ nproc` command (part of GNU Coreutils).

Gnuastro can find the number of threads available to your system at run-time (when you execute the program). However, if a value is given provided for the `--numthreads` option, the given value will be used, not the value automatically read from your system, see Section 4.1.4.2 [Operating modes], page 49, and Section 4.2 [Configuration files], page 51, for ways to set a different value. Thus `--numthreads` is the only option with a value that doesn't have to be specified anywhere on the command-line or in the options.

4.3.1 A note on threads

Spinning off threads internally is not necessarily always the most efficient way to run an application. Creating a new thread isn't a cheap operation for the operating system. It is most useful when the input data are fixed and you want the same operation to be done on parts of it. For example one input image to ImageCrop and multiple crops from various parts of it. In this fashion, the image is loaded into memory once, all the crops are divided between the number of threads internally and each thread cuts out those parts which are assigned to it from the same image. On the other hand, if you have multiple images and you want to crop the same region out of all of them, it is much more efficient to set `--numthreads=1` (so no threads spin off) and run ImageCrop multiple times simultaneously, see Section 4.3.2 [How to run simultaneous operations], page 55.

You can check the boost in speed by first running a program on one of the data sets with the maximum number of threads and another time (with everything else the same) and only using one thread. You will notice that the wall-clock time (reported by most programs at their end) in the former is longer than the latter divided by number of physical CPU cores available to your operating system. Asymptotically these two can be equal (most of the time they aren't). So limiting the programs to use only one thread and running them independently on the number of available threads will be more efficient.

Note that the operating system keeps a cache of recently processed data, so usually, the second time you process an identical data set (independent of the number of threads used), you will get faster results. In order to make an unbiased comparison, you have to first clean the system's cache with the following command between the two runs.

```
$ sync; echo 3 | sudo tee /proc/sys/vm/drop_caches
```

SUMMARY: Should I use multiple threads? Depends:

- If you only have **one** data set (image in most cases!), then yes, the more threads you use (with a maximum of the number of threads available to your OS) the faster you will get your results.
- If you want to run the same operation on **multiple** data sets, it is best to set the number of threads to 1 and use GNU Parallel as explained above.

4.3.2 How to run simultaneous operations

There are two approaches to simultaneously execute a program: using GNU Parallel or Make (GNU Make is the most common implementation). The first is very useful when you only want to do one job multiple times and want to get back to your work without actually keeping the command you ran. The second is usually for (very) complicated processes, with lots of dependencies between the different products (for example a data-production pipeline).

GNU Parallel

When you only want to run multiple instances of a command on different threads and get on with the rest of your work, the best method is to use GNU parallel. Surprisingly GNU Parallel is one of the few GNU packages that has no Info documentation but only a Man page, see Section 4.6.4 [Info], page 60. So to see the documentation after installing it please run

```
$ man parallel
```

As an example, let's assume we want to crop a region fixed on the pixels (500, 600) with the default width from all the FITS images in the `./data` directory ending with `sci.fits` to the current directory. To do this, you can run:

```
$ parallel astimgcrop --numthreads=1 --xc=500 --yc=600 ::: \
  ./data/*sci.fits
```

GNU Parallel can help in many more conditions, this is one of the simplest, see the man page for lots of other examples. For absolute beginners: the backslash (\) is only a line breaker to fit nicely in the page. If you type the whole command in one line, you should remove it.

Make

Make is a program built for specifying "targets", "prerequisites" and "recipes". It allows you to define very complicated dependency structures for complicated processes that commonly start off with a large list of inputs and builds them based on the dependencies you define. GNU Make[4] is the most common implementation which (like nearly all GNU programs comes with a wonderful manual[5]). It is very basic and short (the most important part is about 100 pages).

Make has the `--jobs` (`-j`) which allows you to specify the maximum number of jobs that can be done simultaneously. For example a common 4 physical core CPU usually has 8 processing threads. So you can run:

```
$ make -j8
```

[4] `https://www.gnu.org/software/make/`

[5] `https://www.gnu.org/software/make/manual/`

Once the dependency tree for your processes is built, Make will run the independent targets simultaneously.

4.4 Final parameter values, reproduce previous results

The input parameters can be specified in many places, either on the command-line or in at least one of several configuration files, see Section 4.2 [Configuration files], page 51. Therefore, it often happens that before running a program on a certain data set, you want to see the values for the parameters that the program will use after it has read your command-line options and all the configuration files in their correct order. You might also want to save the list with the output so you can reproduce the same results at a later time, this is very important when you want to use your results in a report or paper.

If you call the `--printparams` option, all Gnuastro programs will read your command-line parameters and all the configuration files. If there is no problem (like a missing parameter or a value in the wrong format) and immediately before actually running, the programs will print the full list of parameter names and values sorted and grouped by context and quit. They will also report their version number, the date they were configured on your system and the time they were reported.

As an example, you can give your full command-line options and even the input and output file names and finally just add `-P` to check if all the parameters are finely set. If everything is ok, you can just run the same command (easily retrieved from the bash history, with the top arrow key) and simply remove the last two characters that showed this option.

Since no program will actually start its processing when this option is called, the otherwise mandatory arguments for each program (for example input image or catalog files) are no longer required when you call this option.

In case you want to store the list of parameters for later reproduction of the same results, you can do so with the GNU Bash re-direction tool. For example after you have produced the results you want to store, you can save all the parameters that were used in a file named `parameters.txt` in the following manner. Using shell history you can retrieve the last command you entered and simply add `-P > parameters.txt` to it, for example:

```
$ astimgcrop --racol=2 --deccol=3 IN.fits cat.txt -P > parameters.txt
```

All the parameters along with the extra data explained before will be stored in the plain text `parameters.txt` file through the shell's redirection mechanism (`>`). The output of `--printparams` conforms with the configuration file formats[6]. By taking the following steps, you can use this file as a configuration file to reproduce your results at a later time.

1. Set the file name based on the standard configuration file names, see Section 4.2.1 [Configuration file format], page 52.
2. Later on (when ever you want to re-produce your results), copy the file in the `./.gnuastro/` directory of your current directory.

In this manner, this file will be read as a current directory configuration file and since all the parameters are defined in it, no other configuration file value will be used.

[6] They are both written by the same function.

4.5 Automatic output

All the programs in Gnuastro are designed such that specifying an output file or directory (based on the program context) is optional. The outputs of most programs are automatically found based on the input and what the program does. For example when you are converting a FITS image named `FITSimage.fits` to a JPEG image, the JPEG image will be saved in `FITSimage.jpg`.

Another very important part of the automatic output generation is that all the directory information of the input file name is stripped off of it. This feature can be disabled with the `--keepinputdir` option, see Section 4.1.4 [Common options], page 48. It is the default because astronomical data are usually very large and organized specially with special file names. In some cases, the user might not have write permissions in those directories. In fact, even if the data is stored on your own computer, it is advised to only grant write permissions to the super user or root. This way, you won't accidentally delete or modify your valuable data!

Let's assume that we are working on a report and want to process the FITS images from two projects (ABC and DEF), which are stored in the sub-directories named `ABCproject/` and `DEFproject/` of our top data directory (`/mnt/data`). The following shell commands show how one image from the former is first converted to a JPEG image through Convert-Type and then the objects from an image in the latter project are detected using NoiseChisel. The text after the `#` sign are comments (not typed!).

```
$ pwd                                                  # Current location
/home/usrname/research/report
$ ls                                             # List directory contents
ABC01.jpg
$ ls /mnt/data/ABCproject                                      # Archive 1
ABC01.fits ABC02.fits ABC03.fits
$ ls /mnt/data/DEFproject                                      # Archive 2
DEF01.fits DEF02.fits DEF03.fits
$ astconvertt /mnt/data/ABCproject/ABC02.fits --output=jpg      # Prog 1
$ ls
ABC01.jpg ABC02.jpg
$ astnoisechisel /mnt/data/DEFproject/DEF01.fits                # Prog 2
$ ls
ABC01.jpg ABC02.jpg DEF01_labeled.fits
```

4.6 Getting help

Probably the first time you read this book, it is either in the PDF or HTML formats. These two formats are very convenient for when you are not actually working, but when you are only reading. Later on, when you start to use the programs and you are deep in the middle of your work, some of the details will inevitably be forgotten. Going to find the PDF file (printed or digital) or the HTML webpage is a major distraction.

GNU software have a very unique set of tools for aiding your memory on the command-line, where you are working, depending how much of it you need to remember. In the past, such command-line help was known as "online" help, because they were literally provided to you 'on' the command 'line'. However, nowadays the word "online" refers to something

on the internet, so that term will not be used. With this type of help, you can resume your exciting research without taking your hands off the keyboard.

Another major advantage of such command-line based help routines is that they are installed with the software in your computer, therefore they are always in sync with the executable you are actually running. Three of them are actually part of the executable. You don't have to worry about the version of the book or program. If you rely on external help (a PDF in your personal print or digital archive or HTML from the official webpage) you have to check to see if their versions fit with your installed program.

If you only need to remember the short or long names of the options, `--usage` is advised. If it is what the options do, then `--help` is a great tool. Man pages are also provided for those who are use to this older system of documentation. This full book is also available to you on the command-line in Info format. If none of these seems to resolve the problems, there is a mailing list which enables you to get in touch with experienced Gnuastro users. In the subsections below each of these methods are reviewed.

4.6.1 `--usage`

If you give this option, the program will not run. It will only print a very concise message showing the options and arguments. Everything within square brackets (`[]`) is optional. For example here are the first and last two lines of ImageCrop's `--usage` is shown:

```
$ astimgcrop --usage
Usage: astimgcrop [-Do?IPqSVW] [-d INT] [-h INT] [-r INT] [-w INT]
            [-x INT] [-y INT] [-c INT] [-p STR] [-N INT] [--deccol=INT]
            ....
            [--setusrconf] [--usage] [--version] [--wcsmode]
            [ASCIIcatalog] FITSimage(s).fits
```

There are no explanations on the options, just their short and long names shown separately. After the program name, the short format of all the options that don't require a value (on/off options) is displayed. Those that do require a value then follow in separate brackets, each displaying the format of the input they want, see Section 4.1.3 [Options], page 46. Since all options are optional, they are shown in square brackets, but arguments can also be optional. For example in this example, a catalog name is optional and is only required in some modes. This is a standard method of displaying optional arguments for all GNU software.

4.6.2 `--help`

If the command-line includes this option, the program will not be run. It will print a complete list of all available options along with a short explanation. The options are also grouped by their context. Within each context, the options are sorted alphabetically. Since the options are shown in detail afterwards, the first line of the `--help` output shows the arguments and if they are optional or not, similar to Section 4.6.1 [`--usage`], page 58.

In the `--help` output of all programs in Gnuastro, the options for each program are classified based on context. The first two contexts are always options to do with the input and output respectively. For example input image extensions or supplementary input files for the inputs. The last class of options is also fixed in all of Gnuastro, it shows operating mode options. Most of these options are already explained in Section 4.1.4.2 [Operating modes], page 49.

The help message will sometimes be longer than the vertical size of your terminal. If you are using a graphical user interface terminal emulator, you can scroll the terminal with your mouse, but we promised no mice distractions! So here are some suggestions:

- `Shift + PageUP` to scroll up and `Shift + PageDown` to scroll down. For most help output this should be enough. The problem is that it is limited by the number of lines that your terminal keeps in memory and that you can't scroll by lines, only by whole screens.
- Pipe to `less`. A pipe is a form of shell re-direction. The `less` tool in Unix-like systems was made exactly for such outputs of any length. You can pipe (`|`) the output of any program that is longer than the screen to it and then you can scroll through (up and down) with its many tools. For example:

  ```
  $ astnoisechisel --help | less
  ```

  Once you have gone through the text, you can quit `less` by pressing the `q` key.
- Redirect to a file. This is a less convenient way, because you will then have to open the file in a text editor! You can do this with the shell redirection tool (`>`):

  ```
  $ astnoisechisel --help > filename.txt
  ```

In case you have a special keyword you are looking for in the help, you don't have to go through the full list. GNU Grep is made for this job. For example if you only want the list of options whose `--help` output contains the word "axis" in ImageCrop, you can run the following command:

```
$ astimgcrop --help | grep axis
```

If the output of this option does not fit nicely within the confines of your terminal, GNU does enable you to customize its output through the environment variable `ARGP_HELP_FMT`, you can set various parameters which specify the formatting of the help messages. For example if your terminals are wider than 70 spaces (say 100) and you feel there is too much empty space between the long options and the short explanation, you can change these formats by giving values to this environment variable before running the program with the `--help` output. You can define this environment variable in this manner:

```
$ export ARGP_HELP_FMT=rmargin=100,opt-doc-col=20
```

This will affect all GNU programs using GNU C library's `argp.h` facilities as long as the environment variable is in memory. You can see the full list of these formatting parameters in the "Argp User Customization" part of the GNU C library manual. If you are more comfortable to read the `--help` outputs of all GNU software in your customized format, you can add your customizations (similar to the line above, without the `$` sign) to your `~/.bashrc` file. This is a standard option for all GNU software.

4.6.3 Man pages

Man pages were the Unix method of providing command-line documentation to a program. With GNU Info, see Section 4.6.4 [Info], page 60, the usage of this method of documentation is highly discouraged. This is because Info provides a much more easier to navigate and read environment.

However, some operating systems require a man page for packages that are installed and some people are still used to this method of command line help. So the programs

in Gnuastro also have Man pages which are automatically generated from the outputs of `--version` and `--help` using the GNU help2man program. So if you run

```
$ man programname
```

You will be provided with a man page listing the options in the standard manner.

4.6.4 Info

Info is the standard documentation format for all GNU software. It is a very useful command-line document viewing format, fully equipped with links between the various pages and menus and search capabilities. As explained before, the best thing about it is that it is available for you the moment you need to refresh your memory on any command-line tool in the middle of your work without having to take your hands off the keyboard. This complete book is available in Info format and can be accessed from anywhere on the command-line.

To open the Info format of any installed programs or library on your system which has an Info format book, you can simply run the command below (change `executablename` to the executable name of the program or library):

```
$ info executablename
```

In case you are not already familiar with it, run `$ info info`. It does a fantastic job in explaining all its capabilities its self. It is very short and you will become sufficiently fluent in about half an hour. Since all GNU software documentation is also provided in Info, your whole GNU/Linux life will significantly improve.

Once you've become an efficient navigator in Info, you can go to any part of this book or any other GNU software or library manual, no matter how long it is, in a matter of seconds. It also blends nicely with GNU Emacs (a text editor) and you can search manuals while you are writing your document or programs without taking your hands off the keyboard, this is most useful for libraries like the GNU C library. To be able to access all the Info manuals installed in your GNU/Linux within Emacs, type `Ctrl-H + i`.

To see this whole book from the beginning in Info, you can run

```
$ info gnuastro
```

If you run Info with the particular program executable name, for example `astimgcrop` or `astnoisechisel`:

```
$ info astprogramname
```

you will be taken to the section titled "Invoking ProgramName" which explains the inputs and outputs along with the command-line options for that program. Finally, if you run Info with the official program name, for example ImageCrop or NoiseChisel:

```
$ info ProgramName
```

you will be taken to the top section which introduces the program. Note that in all cases, Info is not case sensitive.

4.6.5 help-gnuastro mailing list

Gnuastro maintains the help-gnuastro mailing list for users to ask any questions related to Gnuastro. The experienced Gnuastro users and some of its developers are subscribed to this mailing list and your email will be sent to them immediately. However, when contacting

this mailing list please have in mind that they are possibly very busy and might not be able to answer immediately.

To ask a question from this mailing list, send a mail to `help-gnuastro@gnu.org`. Anyone can view the mailing list archives at `http://lists.gnu.org/archive/html/help-gnuastro/`. It is best that before sending a mail, you search the archives to see if anyone has asked a question similar to yours. If you want to make a suggestion or report a bug, please don't send a mail to this mailing list. We have other mailing lists and tools for those purposes, see Section 1.7 [Report a bug], page 9, or Section 1.8 [Suggest new feature], page 11.

4.7 Output headers

The output FITS files created by Gnuastro will have the following two keywords: `DATE`, `CFITSIO`, `WCSLIB` and `GNUASTRO`. The first specifies the time in UT of the file being created. The next three specify the versions of CFITSIO, WCSLIB and Gnuastro that was used to make the file. Note that WCSLIB has only recently added the version reporting capability. If you version of WCSLIB doesn't have this capability, it will not be reported. Some basic information about Gnuastro is also printed. The example below shows the last few keywords of one of the outputs of ImageCrop.

```
                / ImageCrop (GNU Astronomy Utilities 0.1) 0.1:
DATE    = ' ... '               / file creation date ( ... )
CFITSIO = '3.37    '            / CFITSIO version.
WCSLIB  = '5.5     '            / WCSLIB version.
GNUASTRO= '0.1     '            / GNU Astronomy Utilities version.
COMMENT GNU Astronomy Utilities 0.1
COMMENT http://www.gnu.org/software/gnuastro/
END
```

5 Extensions and Tables

This chapter documents those Gnuastro programs that don't directly operate on the contents of the data file, they just print information, or convert from different data types. Before working on a FITS file, it is commonly the case that you are not sure how many extensions it has within it and also what each extension is (image, table or blank). In such cases, Header (see Section 5.1 [Header], page 62) can be handy by printing the full header of a FITS file to the terminal for easy inspection.

In other cases you want to use the data in a FITS file in other programs (for example in reports) that don't recognize the FITS format, for example converting a FITS image into a Jpeg, or PDF format image. ConvertType (see Section 5.2 [ConvertType], page 65) was built for such situations. Finally, the FITS format is not just for images, it can also store tables. Binary tables in particular can be very useful in storing very large catalogs or tables compared to a plain text file. Table (see Section 5.3 [Table], page 71) can be used to choose certain table columns in a FITS table and see them as a human readable output on the terminal, or to save them in another plain text or FITS table.

5.1 Header

The FITS standard requires each extension of a FITS file to have a header, giving basic information about what is in that extension. Each line in the header is for one keyword, specifying its name, value and a short comment string. Besides the basic information, the headers also contain vital information about the data, how they were processed, the instrument specifications that took the image and also the World Coordinate System that is used to translate pixel coordinates to sky or spectrum coordinates on the image or table.

5.1.1 Invoking Header

Header can print or manipulate the header information in an extension of an astronomical data file. The executable name is `astheader` with the following general template

```
$ astheader [OPTION...] ASTRdata
```

One line examples:

```
$ astheader image.fits
$ astheader --update=OLDKEY,153.034,"Old keyword comment"
$ astheader --remove=COMMENT --comment="Anything you like ;-)."
$ astheader --write=MYKEY1,20.00,"An example keyword" --write=MYKEY2,fd
```

If no keyword modification options are given, the full header of the given HDU will be printed on the command-line. If any of the keywords are to be modified, the headers of the input file will be changed. If you want to keep the original FITS file, it is easiest to create a copy first and then run Header on that. In the FITS standard, keywords are always uppercase. So case does not matter in the input or output keyword names you specify.

Most of the options can accept multiple instances in one command. For example you can add multiple keywords to delete by calling delete multiple times, since repeated keywords are allowed, you can even delete the same keyword multiple times. The action of such options will start from the top most keyword.

FITS STANDARD KEYWORDS: Some header keywords are necessary for later operations on a FITS file, for example BITPIX or NAXIS, see the FITS standard for their full list. If you modify (for example remove or rename) such keywords, the FITS file extension might not be usable any more. Also be careful for the world coordinate system keywords, if you modify or change their values, any future world coordinate system (like RA and Dec) measurements on the image will also change.

PRECEDENCE: The order of operations are as follows. Note that while the order within each class of actions is preserved, the order of individual actions is not. So irrespective of what order you called `--delete` and `--update`. First all the delete operations are going to take effect then the update operations.

1. `--delete`
2. `--rename`
3. `--update`
4. `--write`
5. `--asis`
6. `--history`
7. `--comment`
8. `--date`

All possible syntax errors will be reported before the keywords are actually written. FITS errors during any of these actions will be reported, but Header won't stop until all the operations are complete. If `quitonerror` is called, then Header will immediately stop upon the first error.

If only a certain set of header keywords are desired, it is easiest to pipe the output of Header to GNU Grep. Grep is a very powerful and advanced tool to search strings which is precisely made for such situations. For example if you only want to check the size of an image FITS HDU, you can run:

```
$ astheader input.fits | grep NAXIS
```

The options particular to Header can be seen below. See Section 4.1.4 [Common options], page 48, for a list of the options that are common to all Gnuastro programs, they are not repeated here.

`-a`
`--asis` (`=STR`) Write the string within this argument exactly into the FITS file header with no modifications. If it does not conform to the FITS standards, then it might cause trouble, so please be very careful with this option. If you want to define the keyword from scratch, it is best to use the `--write` option (see below) and let CFITSIO worry about the standards. The best way to use this option is when you want to add a keyword from one FITS file to another unchanged and untouched. Below is an example of such a case that can be very useful sometimes:

```
$ key=$(astheader firstimage.fits | grep KEYWORD)
$ astheader --asis="$key" secondimage.fits
```

In particular note the double quotation signs (`"`) around the reference to the `key` shell variable (`$key`), since FITS keywords usually have lots of space characters, if this variable is not enclosed within double quotation marks, the shell will only give the first word in the full keyword to this option, which will definitely be a non-standard FITS keyword and will make it hard to work on the file afterwords. See the "Quoting" section of the GNU Bash manual for more information if your keyword has the special characters `$`, `‘`, or `\`.

`-d`
`--delete` (`=STR`) Delete one instance of the desired keyword. Multiple instances of `--delete` can be given (possibly even for the same keyword). All keywords given will be removed from the headers in the opposite order (last given keyword will be deleted first). If the keyword doesn't exist, Header will give a warning and return with a non-zero value, but will not stop.

`-r`
`--rename` (`=STR`) Rename a keyword to a new value. The old name and the new name should be separated by either a comma (`,`) or a space character. Note that if you use a space character, you have to put the value to this option within double quotation marks (`"`) so the space character is not interpreted as an option separator. Multiple instances of `--rename` can be given in one command. The keywords will be renamed in the specified order.

`-u`
`--update` (`=STR`) Update a keyword, its value, its comments and its units all defined separately. If there are multiple instances of the keyword in the header, they will be changed from top to bottom (with multiple `--update` options).

The format of the values to this option can best be specified with an example:

```
--update=KEYWORD,value,"comments for this keyword",unit
```

The value can be any numerical or string value. Other than the `KEYWORD`, all the other values are optional. To leave a given token empty, follow the preceding comma (`,`) immediately with the next. If any space character is present around the commas, it will be considered part of the respective token. So if more than one token has space characters within it, the safest method to specify a value to this option is to put double quotation marks around each individual token that needs it. Note that without double quotation marks, space characters will be seen as option separators and can lead to undefined behavior.

`-w`
`--write` (`=STR`) Write a keyword to the header. For the possible value input formats, comments and units for the keyword, see the `--update` option above.

`-H`
`--history`
(`=STR`) Add a `HISTORY` keyword to the header. The string given to this keyword will be separated into multiple keyword cards if it is longer than 70 characters. With each run only one value for the `--history` option will be read. If there are multiple, it is the last one.

`-c`
`--comment`
(`=STR`) Add a `COMMENT` keyword to the header. Similar to the explanation for `--history` above.

`-t`
`--date` Put the current date and time in the header. If the `DATE` keyword already exists in the header, it will be updated.

`-Q`
`--quitonerror`
Quit if any of the operations above are not successful. By default if an error occurs, Header will warn the user of the faulty keyword and continue with the rest of actions.

5.2 ConvertType

The formats of astronomical data were defined mainly for archiving and processing. In other situations, the data might be useful in other formats. For example, when you are writing a paper or report or if you are making slides for a talk, you can't use a FITS image. Other image formats should be used. In other cases you might want your pixel values in a table format as plain text for input to other programs that don't recognize FITS, or to include as a table in your report. ConvertType is created for such situations. The various types will increase with future updates and based on need.

The conversion is not only one way (from FITS to other formats), but two ways (except the EPS and PDF formats). So you can convert a JPEG image or text file into a FITS image. Basically, other than EPS, you can use any of the recognized formats as different color channel inputs to get any of the recognized outputs. So before explaining the options and arguments, first a short description of the recognized files types will be given followed a short introduction to digital color.

5.2.1 Recognized file types

The various standards and the file name extensions recognized by ConvertType are listed below.

FITS or IMH
Astronomical data are commonly stored in the FITS format (and in older data sets in IRAF `.imh` format), a list of file name suffixes which indicate that the file is in this format is given in Section 4.1.2 [Arguments], page 46.

Each extension of a FITS image only has one value per pixel, so when used as input, each input FITS image contributes as one color channel. If you want multiple extensions in one FITS file for different color channels, you have to repeat the file name multiple times and use the `--hdu`, `--hdu2`, `--hdu3` or `--hdu4` options to specify the different extensions.

JPEG The JPEG standard was created by the Joint photographic experts group. It is currently one of the most commonly used image formats. Its major advantage is the compression algorithm that is defined by the standard. Like the FITS standard, this is a raster graphics format, which means that it is pixelated.

A JPEG file can have 1 (for grayscale), 3 (for RGB) and 4 (for CMYK) color channels. If you only want to convert one JPEG image into other formats, there is no problem, however, if you want to use it in combination with other input files, make sure that the final number of color channels does not exceed four. If it does, then ConvertType will abort and notify you.

The file name endings that are recognized as a JPEG file for input are: `.jpg`, `.JPG`, `.jpeg`, `.JPEG`, `.jpe`, `.jif`, `.jfif` and `.jfi`.

EPS

The Encapsulated PostScript (EPS) format is essentially a one page PostScript file which has a specified size. PostScript also includes non-image data, for example lines and texts. It is a fully functional programming language to describe a document. Therefore in ConvertType, EPS is only an output format and cannot be used as input. Contrary to the FITS or JPEG formats, PostScript is not a raster format, but is categorized as vector graphics.

The Portable Document Format (PDF) is currently the most common format for documents. Some believe that PDF has replaced PostScript and that PostScript is now obsolete. This view is wrong, a PostScript file is an actual plain text file that can be edited like any program source with any text editor. To be able to display its programmed content or print, it needs to pass through a processor or compiler. A PDF file can be thought of as the processed output of the compiler on an input PostScript file. PostScript, EPS and PDF were created and are registered by Adobe Systems.

With these features in mind, you can see that when you are compiling a document with TEX or LATEX, using an EPS file is much more low level than a JPEG and thus you have much greater control and therefore quality. Since it also includes vector graphic lines we also use such lines to make a thin border around the image to make its appearance in the document much better. No matter the resolution of the display or printer, these lines will always be clear and not pixelated. In the future, addition of text might be included (for example labels or object IDs) on the EPS output. However, this can be done better with tools within TEX or LATEX such as PGF/Tikz[1].

If the final input image (possibly after all operations on the flux explained below) is a binary image or only has two colors of black and white (in segmentation maps for example), then PostScript has another great advantage compared to other formats. It allows for 1 bit pixels (pixels with a value of 0 or 1), this can decrease the output file size by 8 times. So if a grayscale image is binary, ConvertType will exploit this property in the EPS and PDF (see below) outputs.

The standard formats for an EPS file are `.eps`, `.EPS`, `.epsf` and `.epsi`. The EPS outputs of ConvertType have the `.eps` suffix.

PDF

As explained above, a PDF document is a static document description format, viewing its result is therefore much faster and more efficient than PostScript. To create a PDF output, ConvertType will make a PostScript page description and convert that to PDF using GPL Ghostscript. The suffixes recognized for a

[1] `http://sourceforge.net/projects/pgf/`

PDF file are: `.pdf`, `.PDF`. If GPL Ghostscript cannot be run on the PostScript file, it will remain and a warning will be printed.

`blank` This is not actually a file type! But can be used to fill one color channel with a blank value. If this argument is given for any color channel, that channel will not be used in the output.

Plain text Plain text files have the advantage that they can be viewed with any text editor or on the command-line. Most programs also support input as plain text files. In ConvertType, if the input arguments do not have any of the extensions listed above for other formats, the input is assumed to be a text file. Each plain text file is considered to contain one color channel. There is no standard output for plain text files.

If any of the extension above is mis-spelled, this will result in the output becoming a plain text file with that (short) name. If this happens, ConvertType will warn you and write the output as a plain text file. If you don't want that warning, set your plain text output file names longer than 5 characters. When converting an image to plain text, consider the fact that if the image is large the number of columns in each line will become very large, possibly making it very hard to open in some text editors.

5.2.2 Color

An image is a two dimensional array of 2 dimensional elements called pixels. If each pixel only has one value, the image is known as a grayscale image and no color is defined. The range of values in the image can be interpreted as shades of any color, it is customary to use shades of black or grayscale. However, to produce the color spectrum in the digital world, several primary colors must be mixed. Therefore in a color image, each pixel has several values depending on how many primary colors were chosen. For example on the digital monitor or color digital cameras, all colors are built by mixing the three colors of Red-Green-Blue (RGB) with various proportions. However, for printing on paper, standard printers use the Cyan-Magenta-Yellow-Key (CMYK, Key=black) color space. Therefore when printing an RGB image, usually a transformation of color spaces will be necessary.

In a colored digital camera, a color image is produced by dividing the pixel's area between three colors (filters). However in astronomy due to the intrinsic faintness of most of the targets, the collecting area of the pixel is very important for us. Hence the full area of the pixel is used and one value is stored for that pixel in the end. One color filter is used for the whole image. Thus a FITS image is inherently a grayscale image and no color can be defined for it.

One way to represent a grayscale image in different color spaces is to use the same proportions of the primary colors in each pixel. This is the common way most FITS image converters work: they fill all the channels with the same values. The downside is two fold:

- Three (for RGB) or four (for CMYK) values have to be stored for every pixel, this makes the output file very heavy (in terms of bytes).
- If printing, the printing errors of each color channel can make the printed image slightly more blurred than it actually is.

To solve both these problems, the best way is to save the FITS image into the black channel of the CMYK color space. In the RGB color space all three channels have to be

used. The JPEG standard is the only common standard that accepts CMYK color space, that is why currently only the JPEG standard is included and not the PNG standard for example.

The JPEG and EPS standards set two sizes for the number of bits in each channel: 8-bit and 12-bit. The former is by far the most common and is what is used in ConvertType. Therefore, each channel should have values between 0 to $2^8 - 1 = 255$. From this we see how each pixel in a grayscale image is one byte (8 bits) long, in an RGB image, it is 3bytes long and in CMYK it is 4bytes long. But thanks to the JPEG compression algorithms, when all the pixels of one channel have the same value, that channel is compressed to one pixel. Therefore a Grayscale image and a CMYK image that has only the K-channel filled are approximately the same file size.

5.2.3 Invoking ConvertType

ConvertType will convert any recognized input file type to any specified output type. The executable name is `astconvertt` with the following general template

```
$ astconvertt [OPTION...] InputFile [InputFile2] ... [InputFile4]
```

One line examples:

```
$ astconvertt M31.fits --output=pdf
$ astconvertt galaxy.jpg -ogalaxy.fits
$ astconvertt f1.txt f2.txt f3.fits -o.jpg
$ astconvertt M31_r.fits M31_g.fits blank -oeps
```

The file type of the output will be specified with the (possibly complete) file name given to the `--output` option, which can either be given on the command-line or in any of the configuration files (see Section 4.2 [Configuration files], page 51). Note that if the output suffix is not recognized, it will default to plain text format, see Section 5.2.1 [Recognized file types], page 65.

The order of multiple input files is important. After reading the input file(s) the number of color channels in all the inputs will be used to define which color space is being used for the outputs and how each color channel is interpreted. Note that one file might have more than one color channel (for example in the JPEG format). If there is one color channel the output is grayscale, if three input color channels are given they are respectively considered to be the red, green and blue color channels and if there are four color channels they are respectively considered to be cyan, magenta, yellow and black.

The value to `--output` (or `-o`) can be either a full file name or just the suffix of the desired output format. In the former case, that same name will be used for the output. In the latter case, the name of the output file will be set based on the automatic output guidelines, see Section 4.5 [Automatic output], page 57. Note that the suffix name can optionally start a `.` (dot), so for example `--output=.jpg` and `--output=jpg` are equivalent. Be careful that if you want your output in plain text, you have to give the full file name. So if `-otxt` or `--output=.txt` are given, the output file will be named `txt` or `.txt` (the latter will be a hidden file!).

Besides the common set of options explained in Section 4.1.4 [Common options], page 48, the options to ConvertType can be classified into input, output and flux related options. The majority of the options are to do with the flux range. Astronomical data usually have a

very large dynamic range (difference between maximum and minimum value) and different subjects might be better demonstrated with a limited flux range.

Input:

`--hdu2` If the second input file is a FITS file, the value to this option will be used to specify which HDU will be used. Note that for the first file, the (`--hdu` or `-h` in the common options is used)

`--hdu3` The HDU of the third input FITS file.

`--hdu4` The HDU of the fourth input FITS file.

Output:

`-w`
`--widthincm`
(`=FLT`) The width of the output in centimeters. This is only relevant for those formats that accept such a width (not plain text for example). For most digital purposes, the number of pixels is far more important than the value to this parameter because you can adjust the absolute width (in inches or centimeters) in your document preparation program.

`-b`
`--borderwidth`
(`=INT`) The width of the border to be put around the EPS and PDF outputs in units of PostScript points. There are 72 or 28.35 PostScript points in an inch or centimeter respectively. In other words, there are roughly 3 PostScript points in every millimeter. If you are planning on adding a border, its significance is highly correlated with the value you give to the `--widthincm` parameter.

Unfortunately in the document structuring convention of the PostScript language, the "bounding box" has to be in units of PostScript points with no fractions allowed. So the border values only have to be specified in integers. To have a final border that is thinner than one PostScript point in your document, you can ask for a larger width in ConvertType and then scale down the output EPS or PDF file in your document preparation program. For example by setting `width` in your `includegraphics` command in TEX or LATEX. Since it is vector graphics, the changes of size have no effect on the quality of your output quality (pixels don't get different values).

`-x`
`--hex` Use Hexadecimal encoding in creating EPS output. By default the ASCII85 encoding is used which provides a much better compression ratio. When converted to PDF (or included in TEX or LATEX which is finally saved as a PDF file), an efficient binary encoding is used which is far more efficient than both of them. The choice of EPS encoding will thus have no effect on the final PDF.

So if you want to transfer your EPS files (for example if you want to submit your paper to arXiv or journals in PostScript), their storage might become important if you have large images or lots of small ones. By default ASCII85 encoding is used which offers a much better compression ratio (nearly 40 percent) compared to Hexadecimal encoding.

`-u`
`--quality`
(`=INT`) The quality (compression) of the output JPEG file with values from 0 to 100 (inclusive). For other formats the value to this option is ignored. Note that only in grayscale (when one input color channel is given) will this actually be the exact quality (each pixel will correspond to one input value). If it is in color mode, some degradation will occur. While the JPEG standard does support loss-less graphics, it is not commonly supported.

Flux range:

`-c`
`--change` (`=STR`) Change pixel values with the following format `"from1:to1, from2:to2,..."`. This option is very useful in displaying labeled pixels (not actual data images which have noise) like segmentation maps. In labeled images, usually a group of pixels have a fixed integer value. With this option, you can manipulate the labels before the image is displayed to get a better output for print or to emphasize on a particular set of labels and ignore the rest. The labels in the images will be changed in the same order given. By default first the pixel values will be converted then the pixel values will be truncated (see `--fluxlow` and `--fluxhigh`).

You can use any number for the values irrespective of your final output, your given values are stored and used in the double precision floating point format. So for example if your input image has labels from 1 to 20000 and you only want to display those with labels 957 and 11342 then you can run ConvertType with these options:

```
$ astconvertt --change=957:50000,11342:50001 --fluxlow=5e4 \
   --fluxhigh=1e5 segmentationmap.fits --output=jpg
```

While the output JPEG format is only 8 bit, this operation is done in an intermediate step which is stored in double precision floating point. The pixel values are converted to 8-bit after all operations on the input fluxes have been complete. By placing the value in double quotes you can use as many spaces as you like for better readability.

`-C`
`--changeaftertrunc`
Change pixel values (with `--change`) after truncation of the flux values, by default it is the opposite.

`-L`
`--fluxlow`
(`=FLT`) The minimum flux (pixel value) to display in the output image, any pixel value below this value will be set to this value in the output. If the value to this option is the same as `--fluxmax`, then no flux truncation will be applied. Note that when multiple channels are given, this value is used for all the color channels.

`-H`
`--fluxhigh`
(`=FLT`) The maximum flux (pixel value) to display in the output image, see `--fluxlow`.

`-m`
`--maxbyte`
(`=INT`) This is only used for the JPEG and EPS output formats which have an 8-bit space for each channel of each pixel. The maximum value in each pixel can therefore be $2^8 - 1 = 255$. With this option you can change (decrease) the maximum value. By doing so you will decrease the dynamic range. It can be useful if you plan to use those values for other purposes.

`-i`
`--flminbyte`
(`=INT`) If the lowest pixel value in the input channels is larger than the value to `--fluxlow`, then that input value will be redundant. In some situations it might be necessary to set the minimum byte value (0) to correspond to that flux even if the data do not reach that value. With this option you can do this. Note that if the minimum pixel value is smaller than `--fluxlow`, then this option is redundant.

`-a`
`--fhmaxbyte`
(`=INT`) See `--flminbyte`.

`-l`
`--log` Display the logarithm of the input data. This is done after the conversion and flux truncation steps, see above.

`-n`
`--noinvert`
For 8-bit output types (JPEG and EPS for example) the final value that is stored is inverted so white becomes black and vice versa. The reason for this is that astronomical images usually have a very large area of blank sky in them. The result will be that a large are of the image will be black. Therefore, by default the 8-bit values are inverted so the images blend in better with the text in a document.

Note that this behaviour is ideal for grayscale images, if you want a color image, the colors are going to be mixed up. For color images it is best to call this option so the image is not inverted.

5.3 Table

The FITS standard is not just for storing astronomical images, from its early days, it also included tables. Tables are the products of processing astronomical images and spectra. For example in Gnuastro, MakeCatalog will process the defined pixels over an object and produce a catalog (see Section 7.3 [MakeCatalog], page 145). For each identified object, MakeCatalog can print its position on the image or sky, its total brightness and many other

information that is deducible from the given image. Each one of these properties is a column in its output catalog (or table) and for each input object, we have a row.

When there are only a small number of objects (rows) and not too many properties (columns), then a simple plain text file is mainly enough to store, transfer, or even use the produced data. However, to be more efficient in all these aspects, astronomers have defined the FITS binary table standard to store data in a binary (0 and 1) format, not plain text. This can offer major advantages in all those aspects: the file size will be greatly reduced and the reading and writing will be faster (because the RAM and CPU also work in binary).

The FITS standard also defines a standard for ASCII tables, where the data are stored in the human readable ASCII format, but within the FITS file structure. These are mainly useful for keeping ASCII data along with images and possibly binary data as multiple (conceptually related) extensions within a FITS file.

However, this comes at a cost: binary tables are not easily readable by human eyes. There is no standard on how the zero and ones should be interpretted. The Unix-like operating systems have flurished because of a simple fact: communication between the various tools is based on human readible characters[2]. So while the FITS table standards are very beneficial for the tools that recognize them, they are hard to use in the vast majority of available software. This creates limitations for their generic use.

'Table' is Gnuastro's solution to this problem. With Table, FITS tables (ASCII or binary) are directly accessible to the Unix-like operating systems power-users (those working the command-line or shell, see Section 1.6.1 [Command-line interface], page 7). With Table, a FITS table (in binary or ASCII formats) is only one command away from AWK (or any other tool you want to use). Just like a plain text file that you read with the `cat` command. You can pipe the output of Table into any other tool for higher-level processing, see the examples in Section 5.3.1 [Invoking Table], page 72, for some very simple examples.

5.3.1 Invoking Table

Table will convert FITS binary and ASCII tables into other such tables, or print them on the command-line, or save them in a plain text file. Output columns can also be determined by number or regular expression matching of column names. The executable name is `asttable` with the following general template

```
$ asttable [OPTION...] InputFile
```

One line examples:

```
## Get the table column information (name, data type, or units)
$ asttable bintab.fits --information

## Only print those columns which have a name starting with "MAG_"
$ asttable bintab.fits --columns=^MAG_

## Only print the 2nd column, and the third column multiplied by 5
$ asttable bintab.fits | awk '{print $2, 5*$3}'
```

[2] In "The art of Unix programming", Eric Raymond makes this suggestion to programmers: "When you feel the urge to design a complex binary file format, or a complex binary application protocol, it is generally wise to lie down until the feeling passes.". This is a great book and strongly recommended, give it a look if you want to truely enjoy your work/life in this environment.

```
## Only print those rows with a value in the 10th column above 100000
$ asttable bintab.fits | awk '$10>10e5 {print}'

## Sort the output columns by the third column, save output
$ asttable bintab.fits | sort -k3 > output.txt

## Convert a plain text table to a binary FITS table
$ asttable plaintext.txt --output=inbinary.fits
```

Table can accept plain text files, or binary and ASCII FITS table extensions in a FITS file. For the full list of options common to all Gnuastro utilities please see Section 4.1.4 [Common options], page 48. Options can also be stored in directory, user or system-wide configuration files to avoid repeating on the command-line, see Section 4.2 [Configuration files], page 51. Currently all plain text files are processed (and thus printed, or saved to a binary FITS table) as double floating point types. We are still working on this many many more features.

Table does not follow Automatic output that is common in most Gnuastro, see Section 4.5 [Automatic output], page 57. If no value is given to the `--output` option, the desired columns will be printed to the standard output (on the command-line). This feature makes it very useful to directly pipe the output as input to other programs as the examples above demonstrate. Note that the options below which relate to print formatting are only relevant when the output is in human readable format (on the command-line and plain text files), they are ignored when the output is a binary FITS table.

`-i`
`--information`
: Print the information for each column and abort. The information for each column will be printed as a row on the command-line. The column name (if present), units (if present) and datatype will printed. Note that the FITS standard does not require a name or units for columns, only the datatype is mandatory.

`-c`
`--column` (`=STR` or `=INT`) Specify the columns to output for this table. If an integer number is given, the column number will be used (counting from 1). Otherwise the value (a string) will be passed to an internal regular expression processor which will try to match all the columns in the table with the given regular expression. Currently only FITS tables might have column names (it is an optional feature in the FITS standard). There is currently no particular standard to name plain text columns.

This option can be called any number of times with one run of Table. The order of the output columns will be determined by the input order. This option is also not mandatory. If not given, all the input table columns are output.

Regular expressions are a very powerful tool in matching text and since FITS binary tables usually have a large number of columns, this feature can greatly simply the selection of the output columns.

`-I`
`--ignorecase`
Ignore case while matching the column names with the value(s) of the `--column` option. The FITS standard suggests to treat the column names as case insensitive, however it is not a requirement.

`--feg` (`=STR`) Format of printing floating point numbers in non-binary outputs. It can only accept one of the three following values (same as C's `printf`):

- `f`: Print complete floating point value, this is good when the numbers aren't too small, for example 3.286. But it will print all the zeros in 3.2×10^{-15}.
- `e`: Only print in exponential format. This is good for very large or very small numbers, but can make reading the values of more ordinary numbers a little hard.
- `g`: Let the system choose which representation is better for the number.

`--sintwidth`
(`=INT`) The minimum width (number of characters) for printing columns of shorter integer datatypes. The shorter datatypes are considered to be signed and unsigned characters, short integers, integers.

`--lintwidth`
(`=INT`) The minimum width (number of characters) for printing columns of longer datatypes. The longer datatypes are considered to be long and longlong types.

`--floatwidth`
(`=INT`) The minimum width (number of characters) for printing columns of single precision floating point datatypes.

`--doublewidth`
(`=INT`) The minimum width (number of characters) for printing columns of double precision floating point datatypes.

`--strwidth`
(`=INT`) The minimum width (number of characters) for printing columns of strings (given as one column, the FITS standard allows ASCII strings as table elements).

`--floatprecision`
(`=INT`) The number of digits to print after the floating point for single precision floating point numbers.

`--doubleprecision`
(`=INT`) The number of digits to print after the floating point for double precision floating point numbers.

`--fitstabletype`
(`=STR`) The type of FITS table when a FITS file is specified as the output. This option can only have two values: `binary`, or `ascii`. However, currently ASCII FITS table outputs are not yet implemented due to lack of need. If you need it, please get in touch with so we implement it.

6 Image manipulation

Images are one of the major formats of data that is used in astronomy. The functions in this chapter explain the GNU Astronomy Utilities which are provided for their manipulation. For example cropping out a part of a larger image or convolving the image with a given kernel or applying a transformation to it.

6.1 ImageCrop

Astronomical images are often very large, filled with thousands of galaxies. It often happens that you only want a section of the image, or you have a catalog of sources and you want to visually analyze them in small postage stamps. ImageCrop is made to do all these things. When more than one crop is required, ImageCrop will divide the crops between multiple threads to significantly reduce the run time.

Astronomical surveys are usually extremely large. So large in fact, that the whole survey will not fit into a reasonably sized file. Because of this surveys usually cut the final image into separate tiles and store each tile in a file. For example the COSMOS survey's Hubble space telescope, ACS F814W image consists of 81 separate FITS images, with each one having a volume of 1.7 Giga bytes.

Even though the tile sizes are chosen to be large enough that too many galaxies don't fall on the edges of the tiles, inevitably some do and if you simply crop the image of the galaxy from that one tile, you will miss a large area of the surrounding sky (which is essential in estimating the noise). Therefore in its WCS mode, ImageCrop will stitch parts of the tiles that are relevant for a target (with the given width) from all the input images that cover that region into the output. Of course, the tiles have to be present in the list of input files.

ImageCrop also has facilities to crop arbitrary polygons from a set of tiles by stitching the relevant parts of different tiles within the polygon, see `--polygon` in Section 6.1.4 [Invoking ImageCrop], page 78. Alternatively, it can crop out rectangular regions from one image which can come in handy when removing the bias pixels in raw image processing, see Section 6.1.2 [Crop section syntax], page 77.

6.1.1 ImageCrop modes

In order to be as comprehensive as possible, ImageCrop has two major modes of operation listed below.

Image The image mode uses the pixel coordinates. Depending on your command line options, this mode consists of three sub-modes. In image mode, only one image may be input.

- Catalog (multiple crops). Coordinates are read from a text file. The `--xcol` and `--ycol` columns in the catalog are interpreted as the center of a square crop box whose width is specified with the `--iwidth` option in pixels. Since the given pixel has to be on the center, the width has to be an odd number, so if you give an even number for the width, it will be added by one. If a catalog file name is provided (with `--imagemode` activated of course) this mode will be used.
- Center (one crop). The box center is given on the command-line with the `--xc` and `--yc` parameters. The image width is similar to above.

- Section (one crop). You can specify the section of pixels along each axis in the image which you want to be cropped with the `--section` option. See Section 6.1.2 [Crop section syntax], page 77, for a full explanation on the syntax of specifying the desired region.

The latter two cases will only have one crop box. In both cases, ImageCrop will go into the image mode, irrespective of calling `--wcsmode` or the default mode. In the first two cases, since you specify a central pixel, the crop box will be a square with an odd number of pixels on the side, so your desired pixel sits right in the center, see Section 6.1.3 [Blank pixels], page 77, on how to disable this for cases when the box exceeds the image size.

WCS The Right ascension (RA) and Declination (Dec) of the objects in a catalog is used to define the central position of each postage stamp. In this mode, the width (`--wwidth`) is read in units of arc seconds and multiple images (tiles in a survey) can be input. If the objects are closer to the edge of the image than half the required width, other tiles (if they are present in the input files) are used to fill the empty space. The square output cropped box will have an odd number of pixels on the side.

In this mode, the input images do not necessarily have to be the same size, each individual tile can even be smaller than the final crop. In any case, any part of any of the input images which overlaps with the desired region will be used in the crop. Note that if there is an over lap, the pixels from the last input image read are going to be used. The input images all just have to be aligned with the celestial coordinates, see the caution note below.

Similar to the image mode, there are two sub-modes:

- Catalog (multiple crops). Similar to catalog mode in image mode. The RA and Dec column should be specified in the catalog (`--racol` and `--deccol`).
- Center (one crop). You can specify the center of only one crop box (no matter how many input images there are) with the options `--ra` and `--dec`. If it exists in the input images, it will be cropped similar to the catalog mode. If automatic output is triggered (you don't specify a file name for `--output`) and several of the input images are used to stitch and crop the region around the central point, the name of the first input will be used in automatic output, see Section 4.5 [Automatic output], page 57.

CAUTION: In WCS mode, the image has to be aligned with the celestial coordinates, such that the first FITS axis is parallel (opposite direction) to the Right Ascension (RA) while the second FITS axis is parallel to the declination. If these conditions aren't met for an image, ImageCrop will warn you and abort. You have to use other tools to transform the image to the correct directions.

In short, if you don't specify a catalog, you have to specify box coordinates manually on the command-line. When you do specify a catalog, ImageCrop has to be in one of the two major modes (`--imgmode` or `--wcsmode`). Note that the single crop box parameters specified in the sub-modes will not be written to or read from the configuration file, they have to be specified on each execution.

6.1.2 Crop section syntax

When in image mode, one of the methods to crop only one rectangular section from the input image is to use the `--section` option. ImageCrop has a powerful syntax to read the box parameters from a string of characters. If you leave certain parts of the string to be empty, ImageCrop can fill them for you based on the input image sizes.

To define a box, you need the coordinates of two points: the first pixel in the box at (`X1`, `Y1`) and the pixel which is immediately outside of the box (`X2, Y2`), four coordinates in total. The four coordinates can be specified with one string in this format: `X1:X2,Y1:Y2`. It is given to the `--section` option. Therefore, the pixels along the first axis that are $\geq$`X1` and `<X2` will be included in the cropped image. The same goes for the second axis. Note that each different term will be read as an integer, not a float (there are no sub-pixels in ImageCrop, you can use ImageWarp to shift the matrix with any sub-pixel distance, then crop the warped image, see Section 6.4 [ImageWarp], page 109). Also, following the FITS standard, pixel indexes along each axis start from unity(1) not zero(0).

You can omit any of the values and they will be filled automatically. The left hand side of the colon (`:`) will be filled with `1`, and the right side with the image size. So, `2:,:` will include the full range of pixels along the second axis and only those with a first axis index larger than `2` in the first axis. If the colon is omitted for a dimension, then the full range is automatically used. So the same string is also equal to `2:,` or `2:` or even `2`. If you want such a case for the second axis, you should set it to: `,2`.

If you specify a negative value, it will be seen as before the indexes of the image which are outside the image along the bottom or left sides when viewed in SAO ds9. In case you want to count from the top or right sides of the image, you can use an asterisk (`*`). When confronted with a `*`, ImageCrop will replace it with the maximum length of the image in that dimension. So `*-10:*+10,*-20:*+20` will mean that the crop box will be 20×40 pixels in size and only include the top corner of the input image with 3/4 of the image being covered by blank pixels, see Section 6.1.3 [Blank pixels], page 77.

If you feel more comfortable with space characters between the values, you can use as many space characters as you wish, just be careful to put your value in double quotes, for example `--section="5:200, 123:854"`. If you forget, anything after the first space will not be seen by `--section`, because the unquoted space character is one of the characters that separates options on the command-line.

6.1.3 Blank pixels

The cropped box can potentially include pixels that are beyond the image range. For example when a target in the input catalog was very near the edge of the input image. The parts of the cropped image that were not in the input image will be filled with the following two values depending on the data type of the image. In both cases, SAO ds9 will not color code those pixels.

- If the data type of the image is a floating point type (float or double), IEEE NaN (Not a number) will be used.
- For integer types, pixels out of the image will be filled with the value of the `BLANK` keyword in the cropped image header. The value assigned to it is the lowest value possible for that type, so you will probably never need it any way. Only for the

unsigned character type (`BITPIX=8` in the FITS header), the maximum value is used because it is unsigned, the smallest value is zero which is often meaningful.

You can ask for such blank regions to not be included in the output crop image using the `--noblank` option. In such cases, there is no guarantee that the image size of your outputs are what you asked for.

In some survey images, unfortunately they do not use the `BLANK` FITS keyword. Instead they just give all pixels outside of the survey area a value of zero. So by default, when dealing with float or double image types, any values that are 0.0 are also regarded as blank regions. This can be turned off with the `--zeroisnotblank` option.

6.1.4 Invoking ImageCrop

ImageCrop will crop a region from an image. If in WCS mode, it will also stitch parts from separate images in the input files. The executable name is `astimgcrop` with the following general template

```
$ astimgcrop [OPTION...] [ASCIIcatalog] ASTRdata ...
```

One line examples:

```
$ astimgcrop -I catalog.txt image.fits
$ astimgcrop -W catalog.txt /mnt/data/COSMOS/*_drz.fits
$ astimgcrop --section=10:*-10,10:*-10 --hdu=2 image.fits
$ astimgcrop --ra=189.16704 --dec=62.218203 goodsnorth.fits
$ astimgcrop --xc=568.342 --yc=2091.719 --iwidth=200 image.fits
```

ImageCrop has one mandatory argument which is the input image name(s), shown above with `ASTRdata ...`. You can use shell expansions, for example `*` for this if you have lots of images in WCS mode. If the crop box centers are in a catalog, you also have to provide the catalog name as an argument. Alternatively, you have to provide the crop box parameters with command-line options.

When in catalog mode, ImageCrop will run in parallel unless you set `--numthreads=1`, see Section 4.3 [Threads in Gnuastro], page 54. Note that when multiple threads are being used, in verbose mode, the outputs will not be in order. This is because the threads are asynchronous and thus not started in order. This has no effect on the output, see Section 2.1 [Hubble visually checks and classifies his catalog], page 14, for a tutorial on effectively using this feature.

6.1.4.1 ImageCrop options

The options can be classified into the following contexts: Input, Output and operating mode options. Options that are common to all Gnuastro program are listed in Section 4.1.4 [Common options], page 48, and will not be repeated here.

NOTE: The coordinates are in the FITS format. So the first axis is the horizontal axis when viewed in SAO ds9 and the second axis is the vertical. Also in the FITS standard, counting begins from 1 (one) not 0 (zero).

Input image parameters:

`--hstartwcs`
: (`=INT`) Specify the first keyword card (line number) to start finding the input image world coordinate system information. Distortions were only recently included in WCSLIB (from version 5). Therefore until now, different telescope would apply their own specific set of WCS keywords and put them into the image header along with those that WCSLIB does recognize. So now that WCSLIB recognizes most of the standard distortion parameters, they will get confused with the old ones and give completely wrong results. For example in the CANDELS-GOODS South images[1].

  The two `--hstartwcs` and `--hendwcs` are thus provided so when using older datasets, you can specify what region in the FITS headers you want to use to read the WCS keywords. Note that this is only relevant for reading the WCS information, basic data information like the image size are read separately. These two options will only be considered when the value to `--hendwcs` is larger than that of `--hstartwcs`. So if they are equal or `--hstartwcs` is larger than `--hendwcs`, then all the input keywords will be parsed to get the WCS information of the image.

`--hendwcs`
: (`=INT`) Specify the last keyword card to read for specifying the image world coordinate system on the input images. See `--hstartwcs`

Crop box parameters:

`-x`
`--xc`
: (`=FLT`) The first FITS axis value of central position of the crop box in single image mode.

`-y`
`--yc`
: (`=FLT`) The second FITS axis value of the central position of the crop box in single image mode.

`-s`
`--section`
: (`=STR`) Section of the input image which you want to be cropped. See Section 6.1.2 [Crop section syntax], page 77, for a complete explanation on the syntax required for this input.

`-l`
`--polygon`
: (`=STR`) String of crop polygon vertices. Note that currently only convex polygons should be used. In the future we will make it work for all kinds of polygons. Convex polygons are polygons that do not have an internal angle more than 180 degrees. This option can be used both in the image and WCS modes. The rectangular region that completely encompasses the polygon will be kept and all the pixels that are outside of it will be removed.

  The syntax for the polygon vertices is similar to and simpler than that for `--section`. In short, the dimensions of each coordinate are separated by a

[1] `https://archive.stsci.edu/pub/hlsp/candels/goods-s/gs-tot/v1.0/`

comma (`,`) and each vertice is separated by a colon (`:`). You can define as many vertices as you like. If you would like to use space characters between the dimensions and vertices to make them more human-readable, then you have to put the value to this option in double quotation marks.

For example let's assume you want to work on the deepest part of the WFC3/IR images of Hubble Space Telescope eXtreme Deep Field (HST-XDF). According to the webpage (`https://archive.stsci.edu/prepds/xdf/`)[2] the deepest part is contained within the coordinates:

```
[ (53.187414,-27.779152), (53.159507,-27.759633),
  (53.134517,-27.787144), (53.161906,-27.807208) ]
```

They have provided mask images with only these pixels in the WFC3/IR images, but what if you also need to work on the same region in the full resolution ACS images? Also what if you want to use the CANDELS data for the shallow region? Running ImageCrop with `--polygon` will easily pull out this region of the image for you irrespective of the resolution (if you have set the operating mode to WCS mode in your nearest configuration file, there is no need for `--wcsmode`, you may also provide many FITS image or tiles and ImageCrop will stitch them all):

```
$ astimgcrop --wcsmode desired-filter-image(s).fits        \
  --polygon="53.187414,-27.779152 : 53.159507,-27.759633 : \
             53.134517,-27.787144 : 53.161906,-27.807208"
```

`--outpolygon`
: Keep all the regions outside the polygon and mask the inner ones with blank pixels (see Section 6.1.3 [Blank pixels], page 77). This is practically the inverse of the default mode of treating polygons. Note that this option only works when you have only provided one input image. If multiple images are given (in WCS mode), then the full area covered by all the images has to be shown and the polygon excluded. This can lead to a very large area if large surveys like COSMOS are used. So ImageCrop will abort and notify you. In such cases, it is best to crop out the larger region you want, then mask the smaller region with this option.

`-r`
`--ra`
: (`=FLT`) The first FITS axis value of central position of the crop box in single image mode.

`-d`
`--dec`
: (`=FLT`) The second FITS axis value of the central position of the crop box in single image mode.

`-i`
`--xcol`
: (`=INT`) Column number of the first FITS axis position of the box center, starting from zero. In SAO ds9, the first FITS axis is the horizontal axis.

`-j`
`--ycol`
: (`=INT`) Column number of the second FITS axis position of the box center, starting from zero. In SAO ds9, the second FITS axis is the vertical axis.

[2] `https://archive.stsci.edu/prepds/xdf/`

`-a`
`--iwidth` (`=INT`) Width the square box to crop in image mode in units of pixels. In order for the chosen central pixel to be in the center of the cropped image, the final width has to be an odd number, therefore if the value to this option is an even number, the final crop width will be one pixel larger in each dimension. If you want an even sided crop box, use the `--section` option to specify the boundaries of the box, see Section 6.1.2 [Crop section syntax], page 77.

`-f`
`--racol` (`=INT`) Column number of Right Ascension (RA) in the input catalog, starting from zero.

`-g`
`--deccol` (`=INT`) Column number of declination in the input catalog, starting from zero.

`-w`
`--wwidth` (`=FLT`) The width of the crop box in WCS mode in units of arc-seconds.

Output options:

`-c`
`--checkcenter`
(`=INT`) Box size of region in the center of the image to check in units of pixels. This is only used in WCS mode. Because surveys don't often have a clean square or rectangle shape, some of the pixels on the sides of the surveys don't have any data and are commonly filled with zero valued pixels.

If the RA and Dec of any of the targets specified in the catalog fall in such regions, that cropped image will be useless! Therefore with this option, you can specify a width of a small box (3 pixels is often good enough) around the central pixel of the cropped image. If all the pixels in this small box have the value of zero, no cropped image will be created and this object will be flagged in the final log file.

`-p`
`--suffix` (`=STR`) The suffix (or post-fix) of the output files for when you want all the cropped images to have a special ending. One case where this might be helpful is when besides the science images, you want the weight images (or exposure maps, which are also distributed with survey images) of the cropped regions too. So in one run, you can set the input images to the science images and `--suffix=_s.fits`. In the next run you can set the weight images as input and `--suffix=_w.fits`.

`-b`
`--noblank`
Pixels outside of the input image that are in the crop box will not be used. By default they are filled with blank values (depending on type), see Section 6.1.3 [Blank pixels], page 77.

`-z`
`--zeroisnotblank`
In float or double images, it is common to give the value of zero to blank pixels. If the input image type is one of these two types, such pixels will also

be considered as blank. You can disable this behavior with this option, see Section 6.1.3 [Blank pixels], page 77.

Operating mode options:

`-I`
`--imgmode`
Operate in Image mode as described above. This option is only useful when catalog is being provided. If coordinates are given on the command-line, the mode is automatically set based on them.

`-W`
`--wcsmode`
Operate in WCS mode. See explanations for `--imgmode`.

6.1.4.2 ImageCrop output

When a catalog is given, the value of `--output` (see Section 4.1.4 [Common options], page 48) will be seen as the directory to store the output cropped images. In such cases, the outputs will consist of two parts: a variable part (the row number of each target starting from 1) along with a fixed string which you can set with the `--suffix` option. Note that in catalog mode, only one image can be input.

When the crop box is specified on the command-line, the value to `--output` will be used as a file name. If no output is specified or if it is a directory, the output file name will follow the automatic output names of Gnuastro, see Section 4.5 [Automatic output], page 57, for the input image.

The header of each output cropped image will contain the names of the input image(s) it was cut from. If a name is longer than the 70 character space that the FITS standard allows for header keyword values, the name will be cut into several keywords from the nearest slash (`/`). The keywords have the following format: `ICFn_m`. Where `n` is the number of the image used in this crop and `m` is the part of the name. Following the name is another keyword named `ICFnPIX` which shows the pixel range from that input image in the same syntax as Section 6.1.2 [Crop section syntax], page 77.

Once done, a log file will be created in the current directory named `astimgcrop.log`. For each input row, this file will have three columns:

1. The cropped image file name for that row.
2. The number of input images that were used to create that image.
3. A `0` if the central few pixels (value to the `--checkcenter` option) are blank and `1` otherwise.

There are also comments on the top explaining basic information about the run. If the log file cannot be created (for example you don't have write permission in the directory you are running ImageCrop in) or you have specifically asked for no log file (with the `--nolog` option), then a log file will not be created (unless `--individual` is called). The same columns will be printed in verbose mode on the command-line for each row.

6.2 Arithmetic

It is commonly necessary to do arithmetic operations on the astronomical data. For example in the reduction of raw data it is necessary to subtract the Sky value (Section 6.5.1 [Sky value], page 116) from each image image. Later (once the images as warped into a single grid using ImageWarp for example, see Section 6.4 [ImageWarp], page 109), the images can be co-added or the output pixel grid is the average of the pixels of the individual input images. Arithmetic currently uses the reverse polish or postfix notation, see Section 6.2.1 [Reverse polish notation], page 83, for more information on how to run Arithmetic, please see Section 6.2.3 [Invoking Arithmetic], page 86.

6.2.1 Reverse polish notation

The most common notation for arithmetic operations is the infix notation[3] where the operator goes between the two operands, for example $4 + 5$. While the infix notation is the preferred way in most programming languages, currently Arithmetic does not use it since it will require parenthesis which can complicate the implementation of the code. In the near future we do plan to adopt this notation[4], but for the time being (due to time constraints on the developers), Arithmetic uses the postfix or reverse polish notation[5]. The referenced Wikipedia article provides some excellent explanation on this notation but here we will give a short summary for self-sufficiency.

In the postfix notation, the operator is placed after the operands, as we will see below this removes the need to define parenthesis for most ordinary operators. For example, instead of writing `5+6`, we write `5 6 +`. To easily understand how this notation works, you can think of each operand as a node in a first-in-first-out stack. Every time an operator is confronted, it pops the number of operands it needs from the top of the stack (so they don't exist in the stack any more), does its operation and pushes the result back on top of the stack. So if you want the average of 5 and 6, you would write: `5 6 + 2 /`. The operations that are done are:

1. `5` is an operand, so it is pushed to the top of the stack.
2. `6` is an operand, so it is pushed to the top of the stack.
3. `+` is a binary operator, so pull the top two elements of the stack and perform addition on them (the order is `5+6` in the example above). The result is `11`, push it on top of the stack.
4. `2` is an operand so push it onto the top of the stack.
5. `/` is a binary operator, so pull out the top two elements of the stack (top-most is `2`, then `11`) and divide the second one by the first.

In Gnuastro's Arithmetic, the operands can be FITS images or numbers. As you can see, very complicated procedures can be created without the need for parenthesis. Even functions which take an arbitrary number of arguments can be defined in this notation. For example the Postscript language[6] (the programming language in Postscript and compiled into PDF files) uses this notation.

[3] `https://en.wikipedia.org/wiki/Infix_notation`

[4] `https://savannah.gnu.org/task/index.php?13867`

[5] `https://en.wikipedia.org/wiki/Reverse_Polish_notation`

[6] See the EPS and PDF part of Section 5.2.1 [Recognized file types], page 65, for a little more on the Postscript language.

6.2.2 Arithmetic operators

The recognized operators in Arithmetic are listed below. See Section 6.2.1 [Reverse polish notation], page 83, for more on how the operators and operands should be ordered on the command-line. The operands to all operators can be a data array (for example a FITS image) or a number, the output will be an array or number according to the inputs. For example a number multiplied by an array will produce an array. The conditional operators will return pixel, or numerical values of 0 (false) or 1 (true).

`+` Addition, so "`4 5 +`" is equivalent to $4 + 5$.

`-` Subtraction, so "`4 5 -`" is equivalent to $4 - 5$.

`*` Multiplication, so "`4 5 "*"`" is equivalent to 4×5. On the command-line or in scripts, be sure to quote the multiplication sign (for example `"*"`).

`/` Division, so "`4 5 /`" is equivalent to $4/5$.

`abs` Absolute value of first operand, so "`4 abs`" is equivalent to $|4|$.

`pow` First operand to the power of the second, so "`4 5 pow`" is equivalent to 4^5.

`sqrt` The square root of the first operand, so "`4 sqrt`" is equivalent to $\sqrt{4}$.

`log` Natural logarithm of first operand, so "`4 log`" is equivalent to $\ln(4)$.

`log10` Base-10 logarithm of first operand, so "`4 log10`" is equivalent to $\log(4)$.

`minvalue` Minimum value in the top operand on the stack, so "`a.fits min`" will push the the minimum pixel value in this image onto the stack. Therefore this operator is mainly intended for data (for example images), if the top operand is a number, this operator just returns it without any change. So note that when this operator acts on a single image, the output will no longer be an image, but a number.

`maxvalue` Maximum value of first operand, similar to `minvalue`.

`min` If there are several operands (images), each pixel of the output of this operator will be set to the minimum value of all the operands (images) in that pixel. If the operands are all numbers, then their minimum value will be put in the stack. So "`a.fits b.fits c.fits min`" will produce an image with the same size as the inputs but each output pixel is set to the minimum respective pixel value of the three input images. Important notes:

- All the operands must be either images or numbers, they cannot be mixed.
- NaN/blank pixels will be ignored, see Section 6.1.3 [Blank pixels], page 77.
- All available operands on the stack will be pulled for this operator to generate its output. So after calling this operator, there will only be one operand on the stack, see Section 6.2.1 [Reverse polish notation], page 83.

`max` Similar to `min`, but the pixels of the output will contain the maximum of the respective pixels in all operands in the stack.

`average` Similar to `min`, but the pixels of the output will contain the average of the respective pixels in all operands in the stack.

`median` Similar to `min`, but the pixels of the output will contain the median of the respective pixels in all operands in the stack.

`lt` Less than: If the second popped (or left operand in infix notation, see Section 6.2.1 [Reverse polish notation], page 83) value is smaller than the first popped operand, then this function will return a value of 1, otherwise it will return a value of 0. If both operands are images, then all the pixels will be compared with their counterparts in the other image. If only one operand is an image, then all the pixels will be compared with the the single value (number) of the other operand. Finally if both are numbers, then the output is also just one number (0 or 1).

`le` Less or equal: similar to `lt` ('less than' operator), but returning 1 when the second popped operand is smaller or equal to the first.

`gt` Greater than: similar to `lt` ('less than' operator), but returning 1 when the second popped operand is greater than the first.

`ge` Greater or equal: similar to `lt` ('less than' operator), but returning 1 when the second popped operand is larger or equal to the first.

`eq` Equality: similar to `lt` ('less than' operator), but returning 1 when the two popped operands are equal (to double precison floating point accuracy).

`neq` Non-Equality: similar to `lt` ('less than' operator), but returning 1 when the two popped operands are *not* equal (to double precison floating point accuracy).

`isblank` Test for a blank value (see Section 6.1.3 [Blank pixels], page 77). In essence, this is very similar to the conditional operators: the output is either 1 or 0 (see the 'less than' operator above). The difference is that it only needs one operand. Because of the definition of a blank pixel, a blank value is not even equal to itself, so you cannot use the equal operator above to select blank pixels. See the "Blank pixels" box below for more on Blank pixels in Arithmetic.

`and` Logical AND: returns 1 if both operands have a non-zero value and 0 if both are zero. Both operands have to be the same kind: either both images or both numbers.

`or` Logical OR: returns 1 if either one of the operands is non-zero and 0 only when both operators are zero. Both operands have to be the same kind: either both images or both numbers.

`not` Logical NOT: returns 1 when the operand is zero and 0 when the operand is non-zero. The operand can be an image or number, for an image, it is applied to each pixel separately.

`where` Change the input (pixel) value 'where' a certain condition holds. The conditional operators above can be used to define the condition. Three operands are required for `where`. The input format is demonstrated in this simplified example:

```
$ astarithmetic in.fits condition.fits new.fits where
```

The third and second popped operands (`in.fits` and `condition.fits` above, see Section 6.2.1 [Reverse polish notation], page 83) have to be images or all

three operands have to be numbers. In the former case (when the first two are images), the third operand can be an image or a number. The value of any pixel in `in.fits` that corresponds to a non-zero pixel of `condition.fits` will be changed to the value of the same pixel in `new.fits` (or a fixed number if the first popped operand is a number). Commonly you won't be dealing with an actual FITS file of a conditional image. You will probably define the condition in the same run based on some other reference image and use the conditional and logical operators above to make a true/false (or one/zero) image for you internally. For example the case below:

```
$ astarithmetic in.fits reference.fits 100 gt new.fits where
```

Important reminder for multiplication: please quote the `*` sign (for example, `"*"` or `'*'`). Without quotation, the shell will replace `*` with a list of all the files in your working directory which can produce unpredictable outputs. In any case, the input images are not affected, they are only read.

Blank pixels in Arithmetic: Currently all Arithmetic operations are internally done in double precision floating points. So all blank pixels in the image (see Section 6.1.3 [Blank pixels], page 77) will be stored as IEEE NaN values. One particular aspect of NaN values is that by definition they will fail on *any* comparison. Hence both equal and not-equal operators will fail when both their operands are NaN! The only way to select blank pixels is through the `isblank` operator explained above.

One way you can exploit this property of the NaN value to your advantage is when you want a fully zero-valued image (even over the blank pixels) based on an already existing image (with same size and world coordinate system settings). The following command will produce this for you:

```
$ astarithmetic input.fits nan eq --output=all-zeros.fits
```

Note that on the command-line you can write NaN in any case (for example `NaN`, or `NAN` are also acceptable). Reading NaN as a floating point number in Gnuastro isn't case-sensitive.

6.2.3 Invoking Arithmetic

Arithmetic will do pixel to pixel arithmetic operations on the individual pixels of input data and/or numbers. All input data files must have the same dimensions. The general template is:

```
$ astarithmetic [OPTION...] ASTRdata1 [ASTRdata2] OPERATOR ...
```

One line examples:

```
$ astarithmetic 10.32 3.84 - 2.7 pow            # will print 155.329
$ astarithmetic 1 image.fits / --out=inverse.fits
$ astarithmetic image.fits -1 "*" --out=negative.fits
$ astarithmetic image.fits image.fits - --out=skysub.fits          \
                --hdu=0 --hdu=3
$ astarithmetic image1.fits image2.fits + 2 / --out=average.fits
$ astarithmetic image1.fits image2.fits average --out=average.fits
```

```
$ astarithmetic img1.fits img2.fits img3.fits median              \
                -h0 -h1 -h2 --out=median.fits
```

If the output is an image, and the `--output` option is not given, automatic output will use the name of the first FITS image encountered to generate an output file name, see Section 4.5 [Automatic output], page 57. Also, output WCS information will be taken from the first input image encountered. If the output is a single number, that number will be printed in the standard output. See Section 6.2.1 [Reverse polish notation], page 83, for the notation used to mix operands and operators on the command-line.

Currently Arithmetic will convert the input image into double precision floating point arrays for the operations. But there are plans to allow it to also operate on integer (labeled or masked) images with bit-wise operators so mask layers can also be manipulated[7]. Unless otherwise stated for each operator, blank pixels in the input image will automatically be set as blank in the output. To ignore certain pixels (see Section 6.1.3 [Blank pixels], page 77), you can specify a mask image, see Section 6.5.3 [Mask image], page 125. In that case, when reading the first FITS image, all the masked pixels will be changed to a NaN value. Therefore in any further operations with other images, the output for those pixels will always be a NaN (blank pixel value for floating point data type). Currently all the operations in Arithmetic are done in double precision floating point type. However, if none of the input images have this type, the output will be stored as a single precision floating point type.

The hyphen (`-`) can be used both to specify options (see Section 4.1.3 [Options], page 46) and also to specify a negative number which might be necessary in your arithmetic. In order to enable you to do this, Arithmetic will first parse all the input strings and if the first character after a hyphen is a digit, then that hyphen is temporarily replaced by the vertical tab character which is not commonly used. The arguments are then parsed and these strings will not be specified as an option. Then the given arguments are parsed and any vertical tabs are replaced back with a hyphen so they can be read as negative numbers. So as long as the names of the files you want to work on do not start with a vertical tab followed by a digit, there is no problem. An important consequence of this implementation is that you should not write negative fractions like this: `-.3`, instead write them as `-0.3`.

Without any images, Arithmetic will act like a simple calculator and print the resulting output number on the standard output like the first example above. If you really want such calculator operations on the command-line, AWK (GNU AWK is the most common implementation) is much faster, easier and much more powerful. For example, the numerical one-line example above can be done with the following command. In general AWK is a fantastic tool and GNU AWK has a wonderful manual (`https://www.gnu.org/software/gawk/manual/`). So if you confront situations like this a lot or have to work with large text tables/catalogs, be sure to checkout AWK and simplify your life.

```
$ echo "" | awk '{print (10.32-3.84)^2.7}'
155.329
```

See Section 4.1.4 [Common options], page 48, for a review of the options in all Gnuastro programs, the two options related to a mask image are also specified in Section 6.5.3 [Mask image], page 125. Arithmetic just redefines the `--hdu` option as explained below:

[7] `https://savannah.gnu.org/task/?13869`

`-h`
`--hdu` (`=INT`) Unlike most options in Gnuastro (which will ultimately only have one value for this option), Arithmetic allows `--hdu` to be called multiple times and the value of each invocation will be stored separately (for the unlimited number of input images you would like to use).

The order of the values to `--hdu` is very important (if they don't have the same value!). The order is determined by the order that this option is read: first on the command-line (from left to right), then top-down in each configuration file, see Section 4.2.2 [Configuration file precedence], page 52.

If the number of HDUs is less than the number of input images, Arithmetic will abort and notify you. However, if there are more HDUs than FITS images, there is no problem: they will be used in the given order (every time a FITS image comes up on the stack) and the extra HDUs will be ignored in the end. So there is no problem with having extra HDUs in the configuration files and by default several HDUs with a value of `0` are kept in the system-wide configuration file when you install Gnuastro.

6.3 Convolve

On an image, convolution can be thought of as a process to blur or remove the contrast in an image. If you are already familiar with the concept and just want to run Convolve, you can jump to Section 6.3.4 [Convolution kernel], page 106, and Section 6.3.5 [Invoking Convolve], page 107, and skip the lengthy introduction on the basic definitions and concepts of convolution.

There are generally two methods to convolve an image. The first and more intuitive one is in the "spatial domain" or using the actual image pixel values, see Section 6.3.1 [Spatial domain convolution], page 89. The second method is when we manipulate the "frequency domain", or work on the magnitudes of the different frequencies that constitute the image, see Section 6.3.2 [Frequency domain and Fourier operations], page 91. Understanding convolution in the spatial domain is more intuitive and thus recommended if you are just starting to learn about convolution. However, getting a good grasp of the frequency domain is a little more involved and needs some concentration and some mathematical proofs. However, its reward is a faster operation and more importantly a very fundamental understanding of this very important operation.

Convolution of an image will generally result in blurring the image because it mixes pixel values. In other words, if the image has sharp differences in neighboring pixel values[8], those sharp differences will become smoother. This has very good consequences in detection of signal in noise for example. In an actual observed image, the variation in neighboring pixel values due to noise can be very high. But after convolution, those variations will decrease and we have a better hope in detecting the possible underlying signal. Another case where convolution is extensively used is in mock images and modeling in general, convolution can be used to simulate the effect of the atmosphere or the optical system on the mock profiles that we create, see Section 8.1.1.2 [Point Spread Function], page 163. Convolution is a very interesting and important topic in any form of signal analysis (including astronomical

[8] In astronomy, the only major time we confront such sharp borders in signal are cosmic rays. All other sources of signal in an image are already blurred by the atmosphere or the optics of the instrument.

observations). So we have thoroughly[9] explained the concepts behind it in the following sub-sections.

6.3.1 Spatial domain convolution

The pixels in an input image represent different "spatial" positions, therefore when convolution is done only using the actual input pixel values, we name the process as being done in the "Spatial domain". In particular this is in contrast to the "frequency domain" that we will discuss later in Section 6.3.2 [Frequency domain and Fourier operations], page 91. In the spatial domain (and in realistic situations where the image and the convolution kernel don't extend to infinity), convolution is the process of changing the value of one pixel to the *weighted* average of all the pixels in its *neighborhood*.

The 'neighborhood' of each pixel (how many pixels in which direction) and the 'weight' function (how much each neighboring pixel should contribute depending on its position) are given through a second image which is known as a "kernel"[10].

6.3.1.1 Convolution process

In convolution, the kernel specifies the weight and positions of the neighbors of each pixel. To find the convolved value of a pixel, the central pixel of the kernel is placed on that pixel. The values of each overlapping pixel in the kernel and image are multiplied by each other and summed for all the kernel pixels. To have one pixel in the center, the sides of the convolution kernel have to be an odd number. This process effectively mixes the pixel values of each pixel with its neighbors, resulting in a blurred image compared to the sharper input image.

Formally, convolution is one kind of linear 'spatial filtering' in image processing texts. If we assume that the kernel has $2a+1$ and $2b+1$ pixels on each side, the convolved value of a pixel placed at x and y ($C_{x,y}$) can be calculated from the neighboring pixel values in the input image (I) and the kernel (K) from

$$C_{x,y} = \sum_{s=-a}^{a} \sum_{t=-b}^{b} K_{s,t} \times I_{x+s,y+t}.$$

Any pixel coordinate that is outside of the image in the equation above will be considered to be zero. When the kernel is symmetric about its center the blurred image has the same orientation as the original image. However, if the kernel is not symmetric, the image will be affected in the opposite manner, this is a natural consequence of the definition of spatial filtering. In order to avoid this we can rotate the kernel about its center by 180 degrees so the convolved output can have the same original orientation. Technically speaking, only if the kernel is flipped the process is known *Convolution*. If it isn't it is known as *Correlation*.

To be a weighted average, the sum of the weights (the pixels in the kernel) have to be unity. This will have the consequence that the convolved image of an object and unconvolved object will have the same brightness (see Section 8.1.3 [Flux Brightness and magnitude],

[9] A mathematical will certainly consider this explanation is incomplete and inaccurate. However this text is written for an understanding on the operations that are done on a real (not complex, discrete and noisy) astronomical image, not any general form of abstract function

[10] Also known as filter, here we will use 'kernel'.

page 167), which is natural, because convolution should not eat up the object photons, it only disperses them.

6.3.1.2 Edges in the spatial domain

In purely 'linear' spatial filtering (convolution), there are problems on the edges of the input image. Here we will explain the problem in the spatial domain, see Section 6.3.2.10 [Edges in the frequency domain], page 105. The problem originates from the fact that on the edges, in practice[11], the sum of the weights we use on the actual image pixels is not unity. For example, as discussed above, a profile in the center of an image will have the same brightness before and after convolution. However, for a profile on the edge of the image, the brightness (sum of its pixel fluxes within the image, see Section 8.1.3 [Flux Brightness and magnitude], page 167) will not be equal, some of the flux is going to be 'eaten' by the edges.

If you ran `$ make check` on the source files of Gnuastro, you can see the this effect by comparing the `convolve_frequency.fits` with `convolve_spatial.fits` in the `./tests/` directory. In the spatial domain, by default, no assumption will be made about pixels outside of the image or any blank pixels in the image, see Section 6.1.3 [Blank pixels], page 77. The problem explained above will also occur on the sides of blank regions (which might be masked for example). The solution to this edge effect problem is only possible in the spatial domain. We have to discard the assumption that the sum of the kernel pixels is unity during the convolution process[12]. So taking W as the sum of the kernel pixels that used non-blank and in-image pixels, the equation in Section 6.3.1.1 [Convolution process], page 89, will become:

$$C_{x,y} = \frac{\sum_{s=-a}^{a}\sum_{t=-b}^{b} K_{s,t} \times I_{x+s,y+t}}{W}.$$

In this manner, objects which are near the sides of the image or blank pixels will also have the same brightness (within the image) before and after convolution. This correction is applied by default in Convolve when convolving in the spatial domain. To disable it, you can use the `--noedgecorrection` option. In the frequency domain, there is no way to avoid this loss of flux near the edges of the image, see Section 6.3.2.10 [Edges in the frequency domain], page 105.

Note that the edge effect discussed here is different from the one in Section 8.1.2 [If convolving afterwards], page 167. In making mock images we want to simulate a real observation. In a real observation the images of the galaxies on the sides of the CCD are first blurred by the atmosphere and instrument, then imaged. So light from the parts of a galaxy which are immediately outside the CCD will affect the parts of the galaxy which are covered by the CCD. Therefore in modeling the observation, we have to convolve an image that is larger than the input image by exactly half of the convolution kernel. We can hence conclude that this correction for the edges is only useful when working on actual observed images (where we don't have any more data on the edges) and not in modeling.

[11] Because we assumed the overlapping pixels outside the input image have a value of zero.

[12] ofcourse the sum of the kernel pixels still have to be unity.

6.3.2 Frequency domain and Fourier operations

Getting a good grip on the frequency domain is usually not an easy job! So we have decided to give the issue a complete review here. Convolution in the frequency domain (see Section 6.3.2.6 [Convolution theorem], page 98) heavily relies on the concepts of Fourier transform (Section 6.3.2.4 [Fourier transform], page 96) and Fourier series (Section 6.3.2.3 [Fourier series], page 94) so we will be investigating these important operations first. It has become something of a cliché for people to say that the Fourier series "is a way to represent a (wave-like) function as the sum of simple sine waves" (from Wikipedia). However, sines themselves are abstract functions, so this statement really adds no extra layer of physical insight.

Before jumping head-first into the equations and proofs, we will begin with a historical background to see how the importance of frequencies actually roots in our ancient desire to see everything in terms of circles. A short review of how the complex plane should be interpreted is then given. Having paved the way with these two basics, we define the Fourier series and subsequently the Fourier transform. The final aim is to explain discrete Fourier transform, however some very important concepts need to be solidified first: The Dirac comb, convolution theorem and sampling theorem. So each of these topics are explained in their own separate sub-sub-section before going on to the discrete Fourier transform. Finally we revisit (after Section 6.3.1.2 [Edges in the spatial domain], page 90) the problem of convolution on the edges, but this time in the frequency domain. Understanding the sampling theorem and the discrete Fourier transform is very important in order to be able to pull out valuable science from the discrete image pixels. Therefore we have included the mathematical proofs and figures so you can have a clear understanding of these very important concepts.

6.3.2.1 Fourier series historical background

Ever since the ancient times, the circle has been (and still is) the simplest shape for abstract comprehension. All you need is a center point and a radius and you are done. All the points on a circle are at a fixed distance from the center. However, the moment you try to connect this elegantly simple and beautiful abstract construct (the circle) with the real world (for example compute its area or its circumference), things become really hard (ideally, impossible) because the irrational number π gets involved.

The key to understanding the Fourier series (thus the Fourier transform and finally the Discrete Fourier Transform) is our ancient desire to express everything in terms of circles or the most exceptionally simple and elegant abstract human construct. Most people prefer to say the same thing in a more ahistorical manner: to break a function into sines and cosines. As the term "ancient" in the previous sentence implies, Jean-Baptiste Joseph Fourier (1768 – 1830 A.D.) was not the first person to do this. The main reason we know this process by his name today is that he came up with an ingenious method to find the necessary coefficients (radius of) and frequencies ("speed" of rotation on) the circles for any generic (integrable) function.

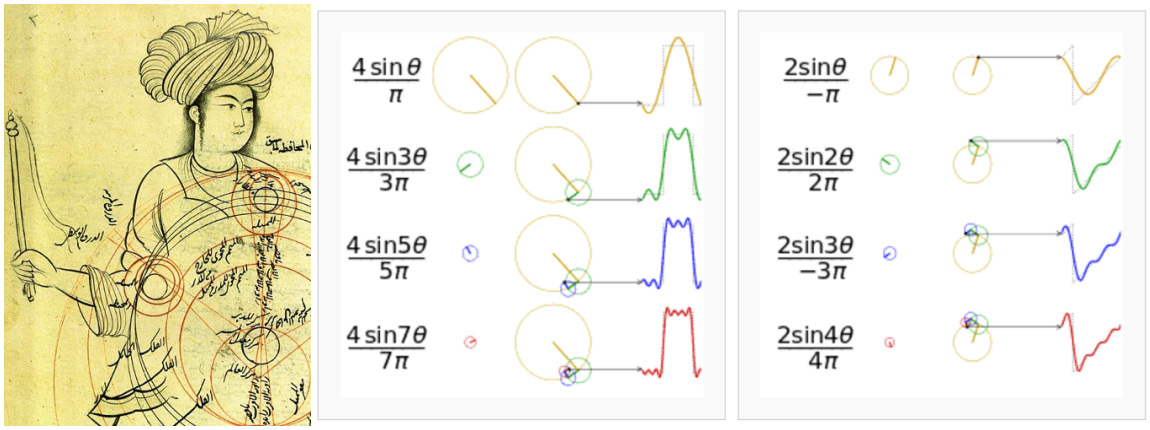


Figure 6.1: Epicycles and the Fourier series. Left: A demonstration of Mercury's epicycles relative to the "center of the world" by Qutb al-Din al-Shirazi (1236 – 1311 A.D.) retrieved from Wikipedia (`https://commons.wikimedia.org/wiki/File:Ghotb2.jpg`). Middle (`https://commons.wikimedia.org/wiki/File:Fourier_series_square_wave_circles_animation.gif`) and Right: How adding more epicycles (or terms in the Fourier series) will approximate functions. The right (`https://commons.wikimedia.org/wiki/File:Fourier_series_sawtooth_wave_circles_animation.gif`) animation is also available.

Like most aspects of mathematics, this process of interpreting everything in terms of circles, began for astronomical purposes. When astronomers noticed that the orbit of Mars and other outer planets, did not appear to be a simple circle (as everything should have been in the heavens). At some point during their orbit, the revolution of these planets would become slower, stop, go back a little (in what is known as the retrograde motion) and then continue going forward again.

The correction proposed by Ptolemy (90 – 168 A.D.) was the most agreed upon. He put the planets on Epicycles or circles whose center itself rotates on a circle whose center is the earth. Eventually, as observations became more and more precise, it was necessary to add more and more epicycles in order to explain the complex motions of the planets[13]. Figure 6.1(Left) shows an example depiction of the epicycles of Mercury in the late 13th century.

Of course we now know that if they had abdicated the Earth from its throne in the center of the heavens and allowed the Sun to take its place, everything would become much simpler and true. But there wasn't enough observational evidence for changing the "professional consensus" of the time to this radical view suggested by a small minority[14]. So the pre-

[13] See the Wikipedia page on "Deferent and epicycle" for a more complete historical review.

[14] Aristarchus of Samos (310 – 230 B.C.) appears to be one of the first people to suggest the Sun being in the center of the universe. This approach to science (that the standard model is defined by consensus) and the fact that this consensus might be completely wrong still applies equally well to our models of particle physics and cosmology today.

Galilean astronomers chose to keep Earth in the center and find a correction to the models (while keeping the heavens a purely "circular" order).

The main reason we are giving this historical background which might appear off topic is to give historical evidence that while such "approximations" do work and are very useful for pragmatic reasons (like measuring the calendar from the movement of astronomical bodies). They offer no physical insight. The astronomers who were involved with the Ptolemaic world view had to add a huge number of epicycles during the centuries after Ptolemy in order to explain more accurate observations. Finally the death knell of this world-view was Galileo's observations with his new instrument (the telescope). So the physical insight, which is what Astronomers and Physicists are interested in (as opposed to Mathematicians and Engineers who just like proving and optimizing or calculating!) comes from being creative and not limiting our selves to such approximations. Even when they work.

6.3.2.2 Circles and the complex plane

Before going onto the derivation, it is also useful to review how the complex numbers and their plane relate to the circles we talked about above. The two schematics in the middle and right of Figure 6.1 show how a 1D function of time can be made using the 2D real and imaginary surface. Seeing the animation in Wikipedia will really help in understanding this important concept. At each point in time, we take the vertical coordinate of the point and use it to find the value of the function at that point in time. Figure 6.2 shows this relation with the axises marked.

Leonhard Euler[15] (1707 – 1783 A.D.) showed that the complex exponential (e^{iv} where v is real) is periodic and can be written as: $e^{iv} = \cos v + isinv$. Therefore $e^{iv+2\pi} = e^{iv}$. Later, Caspar Wessel (mathematician and cartographer 1745 – 1818 A.D.) showed how complex numbers can be displayed as vectors on a plane. Euler's identity might seem counter intuitive at first, so we will try to explain it geometrically (for a more physical insight). On the real-imaginary 2D plane (like the left hand plot in each box of Figure 6.2), multiplying a number by i can be interpretted as rotating the point by 90 degrees (for example the value 3 on the real axis becomes $3i$ on the imaginary axis). On the other hand, $e \equiv \lim_{n\to\infty}(1+\frac{1}{n})^n$, therefore, defining $m \equiv nu$, we get:

$$e^u = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^{nu} = \lim_{n\to\infty}\left(1+\frac{u}{nu}\right)^{nu} = \lim_{m\to\infty}\left(1+\frac{u}{m}\right)^{m}$$

Taking $u \equiv iv$ the result can be written as a generic complex number (a function of v):

$$e^{iv} = \lim_{m\to\infty}\left(1+i\frac{v}{m}\right)^{m} = a(v) + ib(v)$$

For $v = \pi$, a nice geometric animation of going to the limit can be seen on Wikipedia (`https://commons.wikimedia.org/wiki/File:ExpIPi.gif`). We see that $\lim_{m\to\infty} a(\pi) = -1$, while $\lim_{m\to\infty} b(\pi) = 0$, which gives the famus $e^{i\pi} = -1$ equation. The final value is

[15] Other forms of this equation were known before Euler. For example in 1707 A.D. (the year of Euler's birth) Abraham de Moivre (1667 – 1754 A.D.) showed that $(\cos x + i\sin x)^n = \cos(nx) + i\sin(nx)$. In 1714 A.D., Roger Cotes (1682 – 1716 A.D. a colleague of Newton who proofread the second edition of Principia) showed that: $ix = \ln(\cos x + i\sin x)$.

the real number -1, however the distance of the polygon points traversed as $m \to \infty$ is half the circumference of a circle or π, showing how v in the equation above can be interpretted as an angle in units of radians and therefore how $a(v) = cos(v)$ and $b(v) = sin(v)$.

Since e^{iv} is periodic (let's assume with a period of T), it is more clear to write it as $v \equiv \frac{2\pi n}{T}t$ (where n is an integer), so $e^{iv} = e^{i\frac{2\pi n}{T}t}$. The advantage of this notation is that the period (T) is clearly visible and the frequency ($\frac{2\pi n}{T}$, in units of 1/cycle) is defined through the integer n. In this notation, t is in units of "cycle"s.

As we see from the examples in Figure 6.1 and Figure 6.2, for each constituting frequency, we need a respective 'magnitude' or the radius of the circle in order to accurately approximate the desired 1D function. The concepts of "period" and "frequency" are relatively easy to grasp when using temporal units like time because this is how we define them in every-day life. However, in an image (astronomical data), we are dealing with spatial units like distance. Therefore, by one "period" we mean the *distance* at which the signal is identical and frequency is defined as the inverse of that spatial "period". The complex circle of Figure 6.2 can be thought of the Moon rotating about Earth which is rotating around the Sun; so the "Real (signal)" axis shows the Moon's position as seen by a distant observer on the Sun as time goes by. Because of the scalar (not having any direction or vector) nature of time, Figure 6.2 is easier to understand in units of time. When thinking about spatial units, mentally replace the "Time (sec)" axis with "Distance (meters)". Because length has direction and is a vector, visualizing the rotation of the imaginary circle and the advance along the "Distance (meters)" axis is not as simple as temporal units like time.

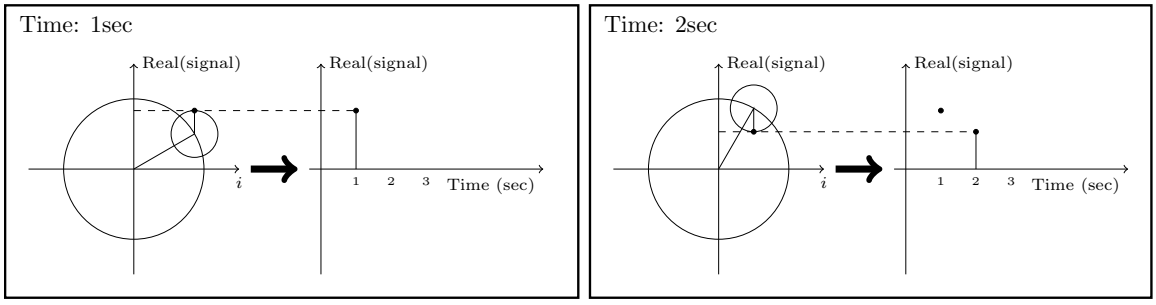


Figure 6.2: Relation between the real (signal), imaginary ($i \equiv \sqrt{-1}$) and time axises at two snapshots of time.

6.3.2.3 Fourier series

In astronomical images, our variable (brightness, or number of photo-electrons, or signal to be more generic) is recorded over the 2D spatial surface of a camera pixel. However to make things easier to understand, here we will assume that the signal is recorded in 1D (assume one row of the 2D image pixels). Also for this section and the next (Section 6.3.2.4 [Fourier transform], page 96) we will be talking about the signal before it is digitized or pixelated. Let's assume that we have the continuous function $f(l)$ which is integrable in the interval $[l_0, l_0 + L]$ (always true in practical cases like images). Take l_0 as the position of the first pixel in the assumed row of the image and L as the width of the image along that row. The units of l_0 and L can be in any spatial units (for example meters) or an angular unit (like radians) multiplied by a fixed distance which is more common.

To approximate $f(l)$ over this interval, we need to find a set of frequencies and their corresponding 'magnitude's (see Section 6.3.2.2 [Circles and the complex plane], page 93). Therefore our aim is to show $f(l)$ as the following sum of periodic functions:

$$f(l) = \sum_{n=-\infty}^{\infty} c_n e^{i\frac{2\pi n}{L}l}$$

Note that the different frequencies ($2\pi n/L$, in units of cycles per meters for example) are not arbitrary. They are all integer multiples of the fundamental frequency of $\omega_0 = 2\pi/L$. Recall that L was the length of the signal we want to model. Therefore, we see that the smallest possible frequency (or the frequency resolution) in the end, depends on the length we observed the signal or L. In the case of each dimension on an image, this is the size of the image in the respective dimension. The frequencies have been defined in this "harmonic" fashion to insure that the final sum is periodic outside of the $[l_0, l_0 + L]$ interval too. At this point, you might be thinking that the sky is not periodic with the same period as my camera's view angle. You are absolutely right! The important thing is that since your camera's observed region is the only region we are "observing" and will be using, the rest of the sky is irrelevant; so we can safely assume the sky is periodic outside of it. However, this working assumption will haunt us later in Section 6.3.2.10 [Edges in the frequency domain], page 105.

The frequencies are thus determined by definition. So all we need to do is to find the coefficients (c_n), or magnitudes, or radii of the circles for each frequency which is identified with the integer n. Fourier's approach was to multiply both sides with a fixed term:

$$f(l)e^{-i\frac{2\pi m}{L}l} = \sum_{n=-\infty}^{\infty} c_n e^{i\frac{2\pi(n-m)}{L}l}$$

where $m > 0$[16]. We can then integrate both sides over the observation period:

$$\int_{l_0}^{l_0+L} f(l)e^{-i\frac{2\pi m}{L}l}dl = \int_{l_0}^{l_0+L} \sum_{n=-\infty}^{\infty} c_n e^{i\frac{2\pi(n-m)}{L}l}dl = \sum_{n=-\infty}^{\infty} c_n \int_{l_0}^{l_0+L} e^{i\frac{2\pi(n-m)}{L}l}dl$$

Both n and m are positive integers. Also, we know that a complex exponential is periodic so after one period (L) it comes back to its starting point. Therefore $\int_{l_0}^{l_0+L} e^{2\pi k/L}dl = 0$ for any $k > 0$. However, when $k = 0$, this integral becomes: $\int_{l_0}^{l_0+T} e^0 dt = \int_{l_0}^{l_0+T} dt = T$. Hence since the integral will be zero for all $n{\neq}m$, we get:

$$\sum_{n=-\infty}^{\infty} c_n \int_{l_0}^{l_0+T} e^{i\frac{2\pi(n-m)}{L}l}dl = Lc_m$$

The origin of the axis is fundamentally an arbitrary position. So let's set it to the start of the image such that $l_0 = 0$. So we can find the "magnitude" of the frequency $2\pi m/L$ within $f(l)$ through the relation:

[16] We could have assumed $m < 0$ and set the exponential to positive, but this is more clear.

$$c_m = \frac{1}{L}\int_0^L f(l)e^{-i\frac{2\pi m}{L}l}dl$$

6.3.2.4 Fourier transform

In Section 6.3.2.3 [Fourier series], page 94, we had to assume that the function is periodic outside of the desired interval with a period of L. Therefore, assuming that $L \to \infty$ will allow us to work with any function. However, with this approximation, the fundamental frequency (ω_0) or the frequency resolution that we discussed in Section 6.3.2.3 [Fourier series], page 94, will tend to zero: $\omega_0 \to 0$. In the equation to find c_m, every m represented a frequency (multiple of ω_0) and the integration on l removes the dependence of the right side of the equation on l, making it only a function of m or frequency. Let's define the following two variables:

$$\omega \equiv m\omega_0 = \frac{2\pi m}{L}$$

$$F(\omega) \equiv Lc_m$$

The equation to find the coefficients of each frequency in Section 6.3.2.3 [Fourier series], page 94, thus becomes:

$$F(\omega) = \int_{-\infty}^{\infty} f(l)e^{-i\omega l}dl.$$

The function $F(\omega)$ is thus the *Fourier transform* of $f(l)$ in the frequency domain. So through this transformation, we can find (analyze) the magnitudes of the constituting frequencies or the value in the frequency space[17] of our spatial input function. The great thing is that we can also do the reverse and later synthesize the input function from its Fourier transform. Let's do it: with the approximations above, multiply the right side of the definition of the Fourier Series (Section 6.3.2.3 [Fourier series], page 94) with $1 = L/L = (\omega_0 L)/(2\pi)$:

$$f(l) = \frac{1}{2\pi}\sum_{n=-\infty}^{\infty} Lc_n e^{\frac{2\pi i n}{L}l}\omega_0 = \frac{1}{2\pi}\sum_{n=-\infty}^{\infty} F(\omega)e^{i\omega l}\Delta\omega$$

To find the right most side of this equation, we renamed ω_0 as $\Delta\omega$ because it was our resolution, $2\pi n/L$ was written as ω and finally, Lc_n was written as $F(\omega)$ as we defined above. Now, as $L \to \infty$, $\Delta\omega \to 0$ so we can write:

$$f(l) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{i\omega l}d\omega$$

[17] As we discussed before, this 'magnitude' can be interpreted as the radius of the circle rotating at this frequency in the epicyclic interpretation of the Fourier series, see Figure 6.1 and Figure 6.2.

Together, these two equations provide us with a very powerful set of tools that we can use to process (analyze) and recreate (synthesize) the input signal. Through the first equation, we can break up our input function into its constituent frequencies and analyze it, hence it is also known as *analysis*. Using the second equation, we can synthesize or make the input function from the known frequencies and their magnitudes. Thus it is known as *synthesis*. Here, we symbolize the Fourier transform (analysis) and its inverse (synthesis) of a function $f(l)$ and its Fourier Transform $F(\omega)$ as $\mathcal{F}[f]$ and $\mathcal{F}^{-1}[F]$.

6.3.2.5 Dirac delta and comb

The Dirac δ (delta) function (also known as an impulse) is the way that we convert a continuous function into a discrete one. It is defined to satisfy the following integral:

$$\int_{-\infty}^{\infty} \delta(l)dl = 1$$

When integrated with another function, it gives that function's value at $l = 0$:

$$\int_{-\infty}^{\infty} f(l)\delta(l)dt = f(0)$$

An impulse positioned at another point (say l_0) is written as $\delta(l - l_0)$:

$$\int_{-\infty}^{\infty} f(l)\delta(l - l_0)dt = f(l_0)$$

The Dirac δ function also operates similarly if we use summations instead of integrals. The Fourier transform of the delta function is:

$$\mathcal{F}[\delta(l)] = \int_{-\infty}^{\infty} \delta(l)e^{-i\omega l}dl = e^{-i\omega 0} = 1$$

$$\mathcal{F}[\delta(l - l_0)] = \int_{-\infty}^{\infty} \delta(l - l_0)e^{-i\omega l}dl = e^{-i\omega l_0}$$

From the definition of the Dirac δ we can also define a Dirac comb (III_P) or an impulse train with infinite impulses separated by P:

$$\mathrm{III}_P(l) \equiv \sum_{k=-\infty}^{\infty} \delta(l - kP)$$

P is chosen to represent "pixel width" later in Section 6.3.2.7 [Sampling theorem], page 100. Therefore the Dirac comb is periodic with a period of P. We have intentionally used a different name for the period of the Dirac comb compared to the input signal's length of observation that we showed with L in Section 6.3.2.3 [Fourier series], page 94. This

difference is highlighted here to avoid confusion later when these two periods are needed together in Section 6.3.2.8 [Discrete Fourier transform], page 102. The Fourier transform of the Dirac comb will be necessary in Section 6.3.2.7 [Sampling theorem], page 100, so let's derive it. By its definition, it is periodic, with a period of P, so the Fourier coefficients of its Fourier Series (Section 6.3.2.3 [Fourier series], page 94) can be calculated within one period:

$$\text{III}_P = \sum_{n=-\infty}^{\infty} c_n e^{i\frac{2\pi n}{P}l}$$

We can now find the c_n from Section 6.3.2.3 [Fourier series], page 94:

$$c_n = \frac{1}{P}\int_{-P/2}^{P/2} \delta(l) e^{-i\frac{2\pi n}{P}l} = \frac{1}{P} \qquad \rightarrow \qquad \text{III}_P = \frac{1}{P}\sum_{n=-\infty}^{\infty} e^{i\frac{2\pi n}{P}l}$$

So we can write the Fourier transform of the Dirac comb as:

$$\mathcal{F}[\text{III}_P] = \int_{-\infty}^{\infty} \text{III}_P e^{-i\omega l} dl = \frac{1}{P}\sum_{n=-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-i(\omega-\frac{2\pi n}{P})l} dl = \frac{1}{P}\sum_{n=-\infty}^{\infty} \delta\left(\omega - \frac{2\pi n}{P}\right)$$

In the last step, we used the fact that the complex exponential is a periodic function, that n is an integer and that as we defined in Section 6.3.2.4 [Fourier transform], page 96, $\omega \equiv m\omega_0$, where m was an integer. The integral will be zero for any ω that is not equal to $2\pi n/P$, a more complete explanation can be seen in Section 6.3.2.3 [Fourier series], page 94. Therefore, while in the spatial domain the impulses had spacing of P (meters for example), in the frequency space, the spacing between the different impulses are $2\pi/P$ cycles per meters.

6.3.2.6 Convolution theorem

The convolution (shown with the $*$ operator) of the two functions $f(l)$ and $h(l)$ is defined as:

$$c(l) \equiv [f*h](l) = \int_{-\infty}^{\infty} f(\tau)h(l-\tau)d\tau$$

See Section 6.3.1.1 [Convolution process], page 89, for a more detailed physical (pixel based) interpretation of this definition. The Fourier transform of convolution ($C(\omega)$) can be written as:

$$C(\omega) = \int_{-\infty}^{\infty} [f*h](l)e^{-i\omega l}dl = \int_{-\infty}^{\infty} f(\tau)\left[\int_{-\infty}^{\infty} h(l-\tau)e^{-i\omega l}dl\right]d\tau$$

To solve the inner integral, let's define $s \equiv l-\tau$, so that $ds = dl$ and $l = s+\tau$ then the inner integral becomes:

$$\int_{-\infty}^{\infty} h(l-\tau)e^{-i\omega l}dl = \int_{-\infty}^{\infty} h(s)e^{-i\omega(s+\tau)}ds = e^{-i\omega\tau}\int_{-\infty}^{\infty} h(s)e^{-i\omega s}ds = H(\omega)e^{-i\omega\tau}$$

where $H(\omega)$ is the Fourier transform of $h(l)$. Substituting this result for the inner integral above, we get:

$$C(\omega) = H(\omega)\int_{-\infty}^{\infty} f(\tau)e^{-i\omega\tau}d\tau = H(\omega)F(\omega) = F(\omega)H(\omega)$$

where $F(\omega)$ is the Fourier transform of $f(l)$. So multiplying the Fourier transform of two functions individually, we get the Fourier transform of their convolution. The convolution theorem also proves a relation between the convolutions in the frequency space. Let's define:

$$D(\omega) \equiv F(\omega) * H(\omega)$$

Applying the inverse Fourier Transform or synthesis equation (Section 6.3.2.4 [Fourier transform], page 96) to both sides and following the same steps above, we get:

$$d(l) = f(l)h(l)$$

Where $d(l)$ is the inverse Fourier transform of $D(\omega)$. We can therefore re-write the two equations above formally as the convolution theorem:

$$\mathcal{F}[f*h] = \mathcal{F}[f]\mathcal{F}[h]$$

$$\mathcal{F}[fh] = \mathcal{F}[f] * \mathcal{F}[h]$$

Besides its usefulness in blurring an image by convolving it with a given kernel, the convolution theorem also enables us to do another very useful operation in data analysis: to match the blur (or PSF) between two images taken with different telescopes/cameras or under different atmospheric conditions. This process is also known as de-convolution. Let's take $f(l)$ as the image with a narrower PSF (less blurry) and $c(l)$ as the image with a wider PSF which appears more blurred. Also let's take $h(l)$ to represent the kernel that should be convolved with the sharper image to create the more blurry image. Above, we proved the relation between these three images through the convolution theorem. But there, we assumed that $f(l)$ and $h(l)$ are known (given) and the convolved image is desired.

In de-convolution, we have $f(l)$ –the sharper image– and $f*h(l)$ –the more blurry image– and we want to find the kernel $h(l)$. The solution is a direct result of the convolution theorem:

$$\mathcal{F}[h] = \frac{\mathcal{F}[f*h]}{\mathcal{F}[f]} \qquad \text{or} \qquad h(l) = \mathcal{F}^{-1}\left[\frac{\mathcal{F}[f*h]}{\mathcal{F}[f]}\right]$$

While this works really nice, it has two problems:

- If $\mathcal{F}[f]$ has any zero values, then the inverse Fourier transform will not be a number!
- If there is significant noise in the image, then the high frequencies of the noise are going to significantly reduce the quality of the final result.

A standard solution to both these problems is the Weiner de-convolution algorithm[18].

6.3.2.7 Sampling theorem

Our mathematical functions are continuous, however, our data collecting and measuring tools are discrete. Here we want to give a mathematical formulation for digitizing the continuous mathematical functions so that later, we can retrieve the continuous function from the digitized recorded input. Assuming that we have a continuous function $f(l)$, then we can define $f_s(l)$ as the 'sampled' $f(l)$ through the Dirac comb (see Section 6.3.2.5 [Dirac delta and comb], page 97):

$$f_s(l) = f(l)\text{III}_P = \sum_{n=-\infty}^{\infty} f(l)\delta(l - nP)$$

The discrete data-element f_k (for example, a pixel in an image), where k is an integer, can thus be represented as:

$$f_k = \int_{-\infty}^{\infty} f_s(l)dl = \int_{-\infty}^{\infty} f(l)\delta(l - kP)dt = f(kP)$$

Note that in practice, our discrete data points are not found in this fashion. Each detector pixel (in an image for example) has an area and averages the signal it receives over that area, not a mathematical point as the Dirac δ function defines. However, as long as the variation in the signal over one detector pixel is not significant, this can be a good approximation. Having put this issue to the side, we can now try to find the relation between the Fourier transforms of the un-sampled $f(l)$ and the sampled $f_s(l)$. For a more clear notation, let's define:

$$F_s(\omega) \equiv \mathcal{F}[f_s]$$

$$D(\omega) \equiv \mathcal{F}[\text{III}_P]$$

Then using the Convolution theorem (see Section 6.3.2.6 [Convolution theorem], page 98), $F_s(\omega)$ can be written as:

[18] `https://en.wikipedia.org/wiki/Wiener_deconvolution`

$$F_s(\omega) = \mathcal{F}[f(l)\text{III}_P] = F(\omega) * D(\omega)$$

Finally, from the definition of convolution and the Fourier transform of the Dirac comb (see Section 6.3.2.5 [Dirac delta and comb], page 97), we get:

$$\begin{aligned} F_s(\omega) &= \int_{-\infty}^{\infty} F(\omega) D(\omega - \mu) d\mu \\ &= \frac{1}{P} \sum_{n=-\infty}^{\infty} \int_{-\infty}^{\infty} F(\omega) \delta\left(\omega - \mu - \frac{2\pi n}{P}\right) d\mu \\ &= \frac{1}{P} \sum_{n=-\infty}^{\infty} F\left(\omega - \frac{2\pi n}{P}\right). \end{aligned}$$

$F(\omega)$ was only a simple function, see Figure 6.3(left). However, from the sampled Fourier transform function we see that $F_s(\omega)$ is the superposition of infinite copies of $F(\omega)$ that have been shifted, see Figure 6.3(right). From the equation, it is clear that the shift in each copy is $2\pi/P$.

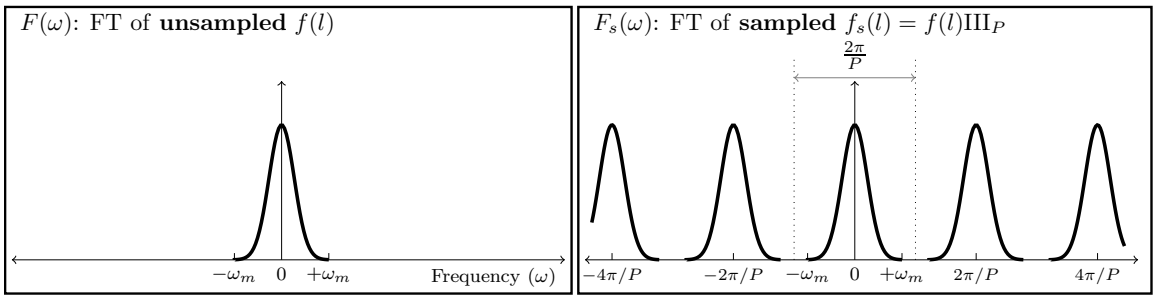


Figure 6.3: Sampling causes infinite repetition in the frequency domain. FT is an abbreviation for 'Fourier transform'. ω_m represents the maximum frequency present in the input. $F(\omega)$ is only symmetric on both sides of 0 when the input is real (not complex). In general $F(\omega)$ is complex and thus cannot be simply plotted like this. Here we have assumed a real Gaussian $f(t)$ which has produced a Gaussian $F(\omega)$.

The input $f(l)$ can have any distribution of frequencies in it. In the example of Figure 6.3(left), the input consisted of a range of frequencies equal to $\Delta\omega = 2\omega_m$. Fortunately as Figure 6.3(right) shows, the assumed pixel size (P) we used to sample this hypothetical function was such that $2\pi/P > \Delta\omega$. The consequence is that each copy of $F(\omega)$ has become completely separate from the surrounding copies. Such a digitized (sampled) data set is thus called *over-sampled*. When $2\pi/P = \Delta\omega$, P is just small enough to finely separate even the largest frequencies in the input signal and thus it is known as *critically-sampled*. Finally if $2\pi/P < \Delta\omega$ we are dealing with an *under-sampled* data set. In an under-sampled data set, the separate copies of $F(\omega)$ are going to overlap and this will deprive us of recovering high constituent frequencies of $f(l)$. The effects of under-sampling in an image with high rates of change (for example a brick wall imaged from a distance) can clearly be visually seen and is known as *aliasing*.

When the input $f(l)$ is composed of a finite range of frequencies, $f(l)$ is known as a *band-limited* function. The example in Figure 6.3(left) was a nice demonstration of such a case: for all $\omega < -\omega_m$ or $\omega > \omega_m$, we have $F(\omega) = 0$. Therefore, when the input function is band-limited and our detector's pixels are placed such that we have critically (or over-) sampled it, then we can exactly reproduce the continuous $f(l)$ from the discrete or digitized samples. To do that, we just have to isolate one copy of $F(\omega)$ from the infinite copies and take its inverse Fourier transform.

This ability to exactly reproduce the continuous input from the sampled or digitized data leads us to the *sampling theorem* which connects the inherent property of the continuous signal (its maximum frequency) to that of the detector (the spacing between its pixels). The sampling theorem states that the full (continuous) signal can be recovered when the pixel size (P) and the maximum constituent frequency in the signal (ω_m) have the following relation[19]:

$$\frac{2\pi}{P} > 2\omega_m$$

This relation was first formulated by Harry Nyquist (1889 – 1976 A.D.) in 1928 and formally proved in 1949 by Claude E. Shannon (1916 – 2001 A.D.) in what is now known as the Nyquist-Shannon sampling theorem. In signal processing, the signal is produced (synthesized) by a transmitter and is received and de-coded (analyzed) by a receiver. Therefore producing a band-limited signal is necessary.

In astronomy, we do not produce the shapes of our targets, we are only observers. Galaxies can have any shape and size, therefore ideally, our signal is not band-limited. However, since we are always confined to observing through an aperture, the aperture will cause a point source (for which $\omega_m = \infty$) to be spread over several pixels. This spread is quantitatively known as the point spread function or PSF. This spread does blur the image which is undesirable; however, for this analysis it produces the positive outcome that there will be a finite ω_m. Though we should caution that any detector will have noise which will add lots of very high frequency (ideally infinite) changes between the pixels. However, the coefficients of those noise frequencies are usually exceedingly small.

6.3.2.8 Discrete Fourier transform

As we have stated several times so far, the input image is a digitized, pixelated or discrete array of values ($f_s(l)$, see Section 6.3.2.7 [Sampling theorem], page 100). The input is not a continuous function. Also, all our numerical calculations can only be done on a sampled, or discrete Fourier transform. Note that $F_s(\omega)$ is not discrete, it is continuous. One way would be to find the analytic $F_s(\omega)$, then sample it at any desired "freq-pixel"[20] spacing. However, this process would involve two steps of operations and computers in particular are not too good at analytic operations for the first step. So here, we will derive a method to directly find the 'freq-pixel'ated $F_s(\omega)$ from the pixelated $f_s(l)$. Let's start with the definition of the Fourier transform (see Section 6.3.2.4 [Fourier transform], page 96):

[19] This equation is also shown in some places without the 2π. Whether 2π is included or not depends on how you define the frequency

[20] We are using the made-up word "freq-pixel" so they are not confused with spatial domain "pixels".

$$F_s(\omega) = \int_{-\infty}^{\infty} f_s(l) e^{-i\omega l} dl$$

From the definition of $f_s(\omega)$ (using x instead of n) we get:

$$\begin{aligned} F_s(\omega) &= \sum_{x=-\infty}^{\infty} \int_{-\infty}^{\infty} f(l)\delta(l - xP) e^{-i\omega l} dl \\ &= \sum_{x=-\infty}^{\infty} f_x e^{-i\omega x P} \end{aligned}$$

Where f_x is the value of $f(l)$ on the point x or the value of the xth pixel. As shown in Section 6.3.2.7 [Sampling theorem], page 100, this function is infinitely periodic with a period of $2\pi/P$. So all we need is the values within one period: $0 < \omega < 2\pi/P$, see Figure 6.3. We want X samples within this interval, so the frequency difference between each frequency sample or freq-pixel is $1/XP$. Hence we will evaluate the equation above on the points at:

$$\omega = \frac{u}{XP} \qquad u = 0, 1, 2, ..., X - 1$$

Therefore the value of the freq-pixel u in the frequency domain is:

$$F_u = \sum_{x=0}^{X-1} f_x e^{-i\frac{ux}{X}}$$

Therefore, we see that for each freq-pixel in the frequency domain, we are going to need all the pixels in the spatial domain[21]. If the input (spatial) pixel row is also X pixels wide, then we can exactly recover the xth pixel with the following summation:

$$f_x = \frac{1}{X} \sum_{u=0}^{X-1} F_u e^{i\frac{ux}{X}}$$

When the input pixel row (we are still only working on 1D data) has X pixels, then it is $L = XP$ spatial units wide. L, or the length of the input data was defined in Section 6.3.2.3 [Fourier series], page 94, and P or the space between the pixels in the input was defined in Section 6.3.2.5 [Dirac delta and comb], page 97. As we saw in Section 6.3.2.7 [Sampling theorem], page 100, the input (spatial) pixel spacing (P) specifies the range of frequencies that can be studied and in Section 6.3.2.3 [Fourier series], page 94, we saw that the length of the (spatial) input, (L) determines the resolution (or size of the freq-pixels) in our discrete Fourier transformed image. Both result from the fact that the frequency domain is the inverse of the spatial domain.

[21] So even if one pixel is a blank pixel (see Section 6.1.3 [Blank pixels], page 77), all the pixels in the frequency domain will also be blank.

6.3.2.9 Fourier operations in two dimensions

Once all the relations in the previous sections have been clearly understood in one dimension, it is very easy to generalize them to two or even more dimentions since each dimension is by definition independent. Previously we defined l as the continuous variable in 1D and the inverse of the period in its direction to be ω. Let's show the second spatial direction with m the the inverse of the period in the second dimension with ν. The Fourier transform in 2D (see Section 6.3.2.4 [Fourier transform], page 96) can be written as:

$$F(\omega,\nu) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(l,m)e^{-i(\omega l+\nu m)}dl$$

$$f(l,m) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(\omega,\nu)e^{i(\omega l+\nu m)}dl$$

The 2D Dirac $\delta(l,m)$ is non-zero only when $l = m = 0$. The 2D Dirac comb (or Dirac brush! See Section 6.3.2.5 [Dirac delta and comb], page 97) can be written in units of the 2D Dirac δ. For most image detectors, the sides of a pixel are equal in both dimentions. So P remains unchanged, if a specific device is used which has non-square pixels, then for each dimension a different value should be used.

$$\mathrm{III}_P(l,m) \equiv \sum_{j=-\infty}^{\infty}\sum_{k=-\infty}^{\infty} \delta(l-jP, m-kP)$$

The Two dimensional Sampling theorem (see Section 6.3.2.7 [Sampling theorem], page 100) is thus very easily derived as before since the frequencies in each dimension are independent. Let's take ν_m as the maximum frequency along the second dimension. Therefore the two dimensional sampling theorem says that a 2D band-limited function can be recovered when the following conditions hold[22]:

$$\frac{2\pi}{P} > 2\omega_m \qquad \text{and} \qquad \frac{2\pi}{P} > 2\nu_m$$

Finally, let's represent the pixel counter on the second dimension in the spatial and frequency domains with y and v respectively. Also let's assume that the input image has Y pixels on the second dimension. Then the two dimensional discrete Fourier transform and its inverse (see Section 6.3.2.8 [Discrete Fourier transform], page 102) can be written as:

$$F_{u,v} = \sum_{x=0}^{X-1}\sum_{y=0}^{Y-1} f_{x,y}e^{-i(\frac{ux}{X}+\frac{vy}{Y})}$$

$$f_{x,y} = \frac{1}{XY}\sum_{u=0}^{X-1}\sum_{v=0}^{Y-1} F_{u,v}e^{i(\frac{ux}{X}+\frac{vy}{Y})}$$

[22] If the pixels are not a square, then each dimension has to use the respective pixel size, but since most imagers have square pixels, we assume so here too

6.3.2.10 Edges in the frequency domain

With a good grasp of the frequency domain, we can revisit the problem of convolution on the image edges, see Section 6.3.1.2 [Edges in the spatial domain], page 90. When we apply the convolution theorem (see Section 6.3.2.6 [Convolution theorem], page 98) to convolve an image, we first take the discrete Fourier transforms (DFT, Section 6.3.2.8 [Discrete Fourier transform], page 102) of both the input image and the kernel, then we multiply them with each other and then take the inverse DFT to construct the convolved image. Of course, in order to multiply them with each other in the frequency domain, the two images have to be the same size, so let's assume that we pad the kernel (it is usually smaller than the input image) with zero valued pixels in both dimensions so it becomes the same size as the input image before the DFT.

Having multiplied the two DFTs, we now apply the inverse DFT which is where the problem is usually created. If the DFT of the kernel only had values of 1 (unrealistic condition!) then there would be no problem and the inverse DFT of the multiplication would be identical with the input. However in real situations, the kernel's DFT has a maximum of 1 (because the sum of the kernel has to be one, see Section 6.3.1.1 [Convolution process], page 89) and decreases something like the hypothetical profile of Figure 6.3. So when multiplied with the input image's DFT, the coefficients or magnitudes (see Section 6.3.2.2 [Circles and the complex plane], page 93) of the smallest frequency (or the sum of the input image pixels) remains unchanged, while the magnitudes of the higher frequencies are significantly reduced.

As we saw in Section 6.3.2.7 [Sampling theorem], page 100, the Fourier transform of a discrete input will be infinitely repeated. In the final inverse DFT step, the input is in the frequency domain (the multiplied DFT of the input image and the kernel DFT). So the result (our output convolved image) will be infinitely repeated in the spatial domain. In order to accurately reconstruct the input image, we need all the frequencies with the correct magnitudes. However, when the magnitudes of higher frequencies are decreased, longer periods (shorter frequencies) will dominate in the reconstructed pixel values. Therefore, when constructing a pixel on the edge of the image, the newly empowered longer periods will look beyond the input image edges and will find the repeated input image there. So if you convolve an image in this fashion using the convolution theorem, when a bright object exists on one edge of the image, its blurred wings will be present on the other side of the convolved image. This is often termed as circular convolution or cyclic convolution.

So as long as we are dealing with convolution in the frequency domain, there is nothing we can do about the image edges. The least we can do is to eliminate the ghosts of the other side of the image. So, we add zero valued pixels to both the input image and the kernel in both dimentions so the image that will be convolved has the a size equal to the sum of both images in each dimension. Of course, the effect of this zero-padding is that the sides of the output convolved image will become dark. To put it another way, the edges are going to drain the flux from nearby objects. But at least it is consistent across all the edges of the image and is predictable. In Convolve, you can see the padded images when inspecting the frequency domain convolution steps with the `--viewfreqsteps` option.

6.3.3 Spatial vs. Frequency domain

With the discussions above it might not be clear when to choose the spatial domain and when to choose the frequency domain. Here we will try to list the benefits of each.

The spatial domain,

- Can correct for the edge effects of convolution, see Section 6.3.1.2 [Edges in the spatial domain], page 90.
- Can operate on blank pixels.
- Can be faster than frequency domain when the kernel is small (in terms of the number of pixels on the sides).

The frequency domain,

- Will be much faster when the image and kernel are both large.

As a general rule of thumb, when working on an image of modeled profiles use the frequency domain and when working on an image of real (observed) objects use the spatial domain (corrected for the edges). The reason is that if you apply a frequency domain convolution to a real image, you are going to loose information on the edges and generally you don't want large kernels. But when you have made the profiles in the image yourself, you can just make a larger input image and crop the central parts to completely remove the edge effect, see Section 8.1.2 [If convolving afterwards], page 167. Also due to oversampling, both the kernels and the images can become very large and the speed boost of frequency domain convolution will significantly improve the processing time, see Section 8.1.1.6 [Oversampling], page 167.

6.3.4 Convolution kernel

All the programs that need convolution will need to be given a convolution kernel file and extension. In most cases (other than Convolve, see Section 6.3 [Convolve], page 88) the kernel file name is optional. However, the extension is necessary and must be specified either on the command-line or at least one of the configuration files (see Section 4.2 [Configuration files], page 51). Within Gnuastro, there are two ways to create a kernel image:

- MakeProfiles: You can use MakeProfiles to create a parametric (based on a radial function) kernel, see Section 8.1 [MakeProfiles], page 162. By default MakeProfiles will make the Gaussian and Moffat profiles in a separate file so you can feed it into any of the programs.
- ConvertType: You can write your own desired kernel into a text file table and convert it to a FITS file with ConvertType, see Section 5.2 [ConvertType], page 65. Just be careful that the kernel has to have an odd number of pixels along its two axises, see Section 6.3.1.1 [Convolution process], page 89. All the programs that do convolution will normalize the kernel internally, so if you choose this option, you don't have to worry about normalizing the kernel. Only within Convolve, there is an option to disable normalization, see Section 6.3.5 [Invoking Convolve], page 107.

The two options to specify a kernel file name and its extension are shown below. These are common between all the programs that will do convolution.

`-k`
`--kernel` (`=STR`) The convolution kernel file name. The `BITPIX` (data type) value of this file can be any standard type and it does not necessarily have to be normalized. Several operations will be done on the kernel image prior to the program's processing:

- It will be converted to floating point type.

- All blank pixels (see Section 6.1.3 [Blank pixels], page 77) will be set to zero.
- It will be normalized so the sum of its pixels equal unity.
- It will be flipped so the convolved image has the same orientation. This is only relevant if the kernel is not circular. See Section 6.3.1.1 [Convolution process], page 89.

`-U`
`--khdu` (`=STR`) The convolution kernel HDU. Although the kernel file name is optional, before running any of the programs, they need to have a value for `--khdu` even if the default kernel is to be used. So be sure to keep its value in at least one of the configuration files (see Section 4.2 [Configuration files], page 51). By default, the system configuration file has a value.

`--fullconvolution`
Ignore the (possible) channels in the mesh grid when doing spatial convolution, see Section 6.5.2 [Tiling an image], page 119. When applied over a mesh grid, spatial convolution will be done independently on each channel. This is necessary when the noise properties of each channel are different and so the pixels should not be mixed. With this option, all channel information is going to be ignored. Currently this option is not deployed for the frequency space convolutions.

6.3.5 Invoking Convolve

Convolve an input image with a known kernel. The general template for Convolve is:

```
$ astconvolve [OPTION...] ASTRdata
```

One line examples:

```
$ astconvolve --kernel=psf.fits mockimg.fits
$ astconvolve --kernel=sharperimage.fits --makekernel\
              blurryimage.fits
```

The only argument accepted by Convolve is an input image file. Some of the options are the same between Convolve and some other Gnuastro programs. Therefore, to avoid repetition, they will not be repeated here. For the full list of options shared by all Gnuastro programs, please see Section 4.1.4 [Common options], page 48. Section 6.5.2.4 [Mesh grid options], page 123, lists all the options related to specifying a mesh grid which is currently only used in spatial convolution. Note that here, no interpolation or smoothing is defined, only channels and the mesh size are important. Section 6.3.4 [Convolution kernel], page 106, lists the the convolution kernel options.

It is also possible to specify a mask image for the input. In that case, see Section 6.5.3 [Mask image], page 125. Here we will only explain the options particular to Convolve. Run Convolve with `--help` in order to see the full list of options Convolve accepts, irrespective of where they are explained in this book.

`--nokernelflip`
Do not flip the kernel after reading it the spatial domain convolution. This can be useful if the flipping has already been applied to the kernel.

`--nokernelnorm`
: Do not normalize the kernel after reading it, such that the sum of its pixels is unity.

`-f`
`--frequency`
: Convolve using discrete Fourier transform in the frequency domain: The Fourier transform of both arrays is first calculated and multiplied. Then the inverse Fourier transform is applied to the product to give the final convolved image.

  For large images, this process will be more efficient than convolving in the spatial domain. However, the edges of the image will loose some flux, see Section 6.3.1.2 [Edges in the spatial domain], page 90.

`-p`
`--spatial`
: Convolve in the spatial domain, see Section 6.3.1.1 [Convolution process], page 89.

`--viewfreqsteps`
: With this option a file with the initial name of the output file will be created that is suffixed with `_freqsteps.fits`, all the steps done to arrive at the final convolved image are saved as extensions in this file. The extensions in order are:

  1. The padded input image. In frequency domain convolution the two images (input and convolved) have to be the same size and both should be padded by zeros.
  2. The padded kernel, similar to the above.
  3. The Fourier spectrum of the forward Fourier transform of the input image. Note that the Fourier transform is a complex operation (and not view able in one image!) So we either have to show the 'Fourier spectrum' or the 'Phase angle'. For the complex number $a + ib$, the Fourier spectrum is defined as $\sqrt{a^2 + b^2}$ while the phase angle is defined as $\arctan(b/a)$.
  4. The Fourier spectrum of the forward Fourier transform of the kernel image.
  5. The Fourier spectrum of the multiplied (through complex arithmetic) transformed images.
  6. The inverse Fourier transform of the multiplied image. If you open it, you will see that the convolved image is now in the center, not on one side of the image as it started with (in the padded image of the first extension). If you are working on a mock image which originally had pixels of precisely 0.0, you will notice that in those parts that your convolved profile(s) did not convert, the values are now $\sim 10^{-18}$, this is due to floating-point round off errors. Therefore in the final step (when cropping the central parts of the image), we also remove any pixel with a value less than 10^{-17}.

`-m`
`--makekernel`
: (`=INT`) If this option is called, Convolve will do de-convolution (see Section 6.3.2.6 [Convolution theorem], page 98). The image specified by the

`--kernel` option is assumed to be the sharper (less blurry) image and the input image is assumed to be the more blurry image. The two images have to be the same size. Some notes to take into account for a good result:

Noise has large frequencies which can make the result less reliable for the higher frequencies of the kernel. So all the frequencies which have a spectrum smaller than 0.005 in the frequency domain are set to zero and not divided. This will cause the wings of the final kernel to be flatter than they would ideally be which will make the convolved image result unreliable. The value given to this option will be used as the maximum radius of the kernel. Any pixel in the final kernel that is larger than this distance from the center will be set to zero.

- Choose a bright (unsaturated) star and use a region box (with ImageCrop for example, see Section 6.1 [ImageCrop], page 75) that is sufficiently above the noise.
- Use ImageWarp (see Section 6.4 [ImageWarp], page 109) to warp the pixel grid so the star's center is exactly on the center of the central pixel in the cropped image. This will certainly slightly degrade the result, however, it is necessary. If there are multiple good stars, you can shift all of them, then normalize them (so the sum of each star's pixels is one) and then take their average to decrease this effect.
- The shifting might move the center of the star by one pixel in any direction, so crop the central pixel of the warped image to have a clean image for the de-convolution.

6.4 ImageWarp

Image warping is the process of mapping the pixels of one image onto a new pixel grid. This process is sometimes known as transformation, however following the discussion of Heckbert 1989[23] we will not be using that term because it can be confused with only pixel value or flux transformations. Here we specifically mean the pixel grid transformation which is better conveyed with 'warp'.

Image wrapping is a very important step in astronomy, both in observational data analysis and in simulating modeled images. In modeling, warping an image is necessary when we want to apply grid transformations to the initial models, for example in simulating gravitational lensing (Radial warpings are not yet included in ImageWarp). Observational reasons for warping an image are listed below:

- **Noise:** Most scientifically interesting targets are inherently faint (have a very low Signal to noise ratio). Therefore one short exposure is not enough to detect such objects that are drowned deeply in the noise. We need multiple exposures so we can add them together and increase the objects' signal to noise ratio. Keeping the telescope fixed on one field of the sky is practically impossible. Therefore very deep observations have to put into the same grid before adding them.
- **Resolution:** If we have multiple images of one patch of the sky (hopefully at multiple orientations) we can warp them to the same grid. The multiple orientations will allow us

[23] Paul S. Heckbert. 1989. *Fundamentals of Texture mapping and Image Warping*, Master's thesis at University of California, Berkely.

to 'guess' the values of pixels on an output pixel grid that has smaller pixel sizes and thus increase the resolution of the output. This process of merging multiple observations is known as Mosaicing.

- **Cosmic rays:** Cosmic rays can randomly fall on any part of an image. If they collide vertically with the camera, they are going to create a very sharp and bright spot that in most cases can be separated easily[24]. However, depending on the depth of the camera pixels, and the angle that a cosmic rays collides with it, it can cover a line-like larger area on the CCD which makes the detection using their sharp edges very hard and error prone. One of the best methods to remove cosmic rays is to compare multiple images of the same field. To do that, we need all the images to be on the same pixel grid.
- **Optical distortion:** (Not yet included in ImageWarp) In wide field images, the optical distortion that occurs on the outer parts of the focal plane will make accurate comparison of the objects at various locations impossible. It is therefore necessary to warp the image and correct for those distortions prior to the analysis.
- **Detector not on focal plane:** In some cases (like the Hubble Space Telescope ACS and WFC3 cameras), the CCD might be tilted compared to the focal plane, therefore the recorded CCD pixels have to be projected onto the focal plane before further analysis.

6.4.1 Warping basics

Let's take $[\,u \quad v\,]$ as the coordinates of a point in the input image and $[\,x \quad y\,]$ as the coordinates of that same point in the output image[25]. The simplest form of coordinate transformation (or warping) is the scaling of the coordinates, let's assume we want to scale the first axis by M and the second by N, the output coordinates of that point can be calculated by

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} Mu \\ Nv \end{bmatrix} = \begin{bmatrix} M & 0 \\ 0 & N \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

Note that these are matrix multiplications. We thus see that we can represent any such grid warping as a matrix. Another thing we can do with this 2×2 matrix is to rotate the output coordinate around the common center of both coordinates. If the output is rotated anticlockwise by θ degrees from the positive (to the right) horizontal axis, then the warping matrix should become:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ucos\theta - vsin\theta \\ usin\theta + vcos\theta \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

We can also flip the coordinates around the first axis, the second axis and the coordinate center with the following three matrices respectively:

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

[24] All astronomical targets are blurred with the PSF, see Section 8.1.1.2 [Point Spread Function], page 163, however a cosmic ray is not and so it is very sharp (it suddenly stops at one pixel).

[25] These can be any real number, we are not necessarily talking about integer pixels here.

The final thing we can do with this definition of a 2×2 warping matrix is shear. If we want the output to be sheared along the first axis with A and along the second with B, then we can use the matrix:

$$\begin{bmatrix} 1 & A \\ B & 1 \end{bmatrix}$$

To have one matrix representing any combination of these steps, you use matrix multiplication, see Section 6.4.2 [Merging multiple warpings], page 112. So any combinations of these transformations can be displayed with one 2×2 matrix:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

The transformations above can cover a lot of the needs of most coordinate transformations. However they are limited to mapping the point $[\,0 \quad 0\,]$ to $[\,0 \quad 0\,]$. Therefore they are useless if you want one coordinate to be shifted compared to the other one. They are also space invariant, meaning that all the coordinates in the image will receive the same transformation. In other words, all the pixels in the output image will have the same area if placed over the input image. So transformations which require varying output pixel sizes like projections cannot be applied through this 2×2 matrix either (for example for the tilted ACS and WFC3 camera detectors on board the Hubble space telescope).

To add these further capabilities, namely translation and projection, we use the homogeneous coordinates. They were defined about 200 years ago by August Ferdinand Möbius (1790 – 1868). For simplicity, we will only discuss points on a 2D plane and avoid the complexities of higher dimensions. We cannot provide a deep mathematical introduction here, interested readers can get a more detailed explanation from Wikipedia[26] and the references therein.

By adding an extra coordinate to a point we can add the flexibility we need. The point $[\,x \quad y\,]$ can be represented as $[\,xZ \quad yZ \quad Z\,]$ in homogeneous coordinates. Therefore multiplying all the coordinates of a point in the homogeneous coordinates with a constant will give the same point. Put another way, the point $[\,x \quad y \quad Z\,]$ corresponds to the point $[\,x/Z \quad y/Z\,]$ on the constant Z plane. Setting $Z = 1$, we get the input image plane, so $[\,u \quad v \quad 1\,]$ corresponds to $[\,u \quad v\,]$. With this definition, the transformations above can be generally written as:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

We thus acquired 4 extra degrees of freedom. By giving non-zero values to the zero valued elements of the last column we can have translation (try the matrix multiplication!). In

[26] `http://en.wikipedia.org/wiki/Homogeneous_coordinates`

general, any coordinate transformation that is represented by the matrix below is known as an affine transformation[27]:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

We can now consider translation, but the affine transform is still spatially invariant. Giving non-zero values to the other two elements in the matrix above gives us the projective transformation or Homography[28] which is the most general type of transformation with the 3×3 matrix:

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

So the output coordinates can be calculated from:

$$x = \frac{x'}{w} = \frac{au + bv + c}{gu + hv + 1} \qquad y = \frac{y'}{w} = \frac{du + ev + f}{gu + hv + 1}$$

Thus with homography we can change the sizes of the output pixels on the input plane, giving a 'perspective'-like visual impression. This can be quantitatively seen in the two equations above. When $g = h = 0$, the denominator is independent of u or v and thus we have spatial invariance. Homography preserves lines at all orientations. A very useful fact about homography is that its inverse is also a homography. These two properties play a very important role in the implementation of this transformation. A short but instructive and illustrated review of affine, projective and also bi-linear mappings is provided in Heckbert 1989[29].

6.4.2 Merging multiple warpings

In Section 6.4.1 [Warping basics], page 110, we saw how one basic warping/transformation can be represented with a 3 by 3 matrix. To make more complex warpings these matrices have to be multiplied through matrix multiplication. However matrix multiplication is not commutative, so the order of the set of matrices you use for the multiplication is going to be very important.

The first warping should be placed as the left-most matrix. The second warping to the right of that and so on. The second transformation is going to occur on the warped coordinates of the first. As an example for merging a few transforms into one matrix, the multiplication below represents the rotation of an image about a point $[\,U \quad V\,]$ anticlockwise from the horizontal axis by an angle of θ. To do this, first we take the origin to $[\,U \quad V\,]$

[27] `http://en.wikipedia.org/wiki/Affine_transformation`

[28] `http://en.wikipedia.org/wiki/Homography`

[29] Paul S. Heckbert. 1989. *Fundamentals of Texture mapping and Image Warping*, Master's thesis at University of California, Berkely. Note that since points are defined as row vectors there, the matrix is the transpose of the one discussed here.

through translation. Then we rotate the image, then we translate it back to where it was initially. These three operations can be merged in one operation by calculating the matrix multiplication below:

$$\begin{bmatrix} 1 & 0 & U \\ 0 & 1 & V \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -U \\ 0 & 1 & -V \\ 0 & 0 & 1 \end{bmatrix}$$

6.4.3 Resampling

A digital image is composed of discrete 'picture elements' or 'pixels'. When a real image is created from a camera or detector, each pixel's area is used to store the number of photoelectrons that were created when incident photons collided with that pixel's surface area. This process is called the 'sampling' of a continuous or analog data into digital data. When we change the pixel grid of an image or warp it as we defined in Section 6.4.1 [Warping basics], page 110, we have to 'guess' the flux value of each pixel on the new grid based on the old grid, or resample it. Because of the 'guessing', any form of warping on the data is going to degrade the image and mix the original pixel values with each other. So if an analysis can be done on an un-warped data image, it is best to leave the image untouched and pursue the analysis. However as discussed in Section 6.4 [ImageWarp], page 109, this is not possible most of the times, so we have to accept the problem and re-sample the image.

In most applications of image processing, it is sufficient to consider each pixel to be a point and not an area. This assumption can significantly speed up the processing of an image and also the simplicity of the code. It is a fine assumption when the signal to noise ratio of the objects are very large. The question will then be one of interpolation because you have multiple points distributed over the output image and you want to find the values at the pixel centers. To increase the accuracy, you might also sample more than one point from within a pixel giving you more points for a more accurate interpolation in the output grid.

However, interpolation has several problems. The first one is that it will depend on the type of function you want to assume for the interpolation. For example you can choose a bi-linear or bi-cubic (the 'bi's are for the 2 dimensional nature of the data) interpolation method. For the latter there are various ways to set the constants[30]. Such functional interpolation functions can fail seriously on the edges of an image. They will also need normalization so that the flux of the objects before and after the warpings are comparable. The most basic problem with such techniques is that they are based on a point while a detector pixel is an area. They add a level of subjectivity to the data (make more assumptions through the functions than the data can handle). For most applications this is fine, but in scientific applications where detection of the faintest possible galaxies or fainter parts of bright galaxies is our aim, we cannot afford this loss. Because of these reasons ImageWarp will not use such interpolation techniques.

ImageWarp will do interpolation based on "pixel mixing"[31] or "area resampling". This is also what the Hubble Space Telescope pipeline calls "Drizzling"[32]. This technique re-

[30] see `http://entropymine.com/imageworsener/bicubic/` for a nice introduction.

[31] For a graphic demonstration see `http://entropymine.com/imageworsener/pixelmixing/`.

[32] `http://en.wikipedia.org/wiki/Drizzle_(image_processing)`

quires no functions, it is thus non-parametric. It is also the closest we can get (make least assumptions) to what actually happens on the detector pixels. The basic idea is that you reverse-transform each output pixel to find which pixels of the input image it covers and what fraction of the area of the input pixels are covered. To find the output pixel value, you simply sum the value of each input pixel weighted by the overlap fraction (between 0 to 1) of the output pixel and that input pixel. Through this process, pixels are treated as an area not as a point (which is how detectors create the image), also the brightness (see Section 8.1.3 [Flux Brightness and magnitude], page 167) of an object will be left completely unchanged.

If there are very high spatial-frequency signals in the image (for example fringes) which vary on a scale smaller than your output image pixel size, pixel mixing can cause ailiasing[33]. So if the input image has fringes, they have to be calculated and removed separately (which would naturally be done in any astronomical application). Because of the PSF no astronomical target has a sharp change in the signal so this issue is less important for astronomical applications, see Section 8.1.1.2 [Point Spread Function], page 163.

6.4.4 Invoking ImageWarp

The general template for invoking ImageWarp is:

```
$ astimgwarp [OPTIONS...] [matrix.txt] InputImage
```

One line examples:

```
$ astimgwarp matrix.txt image.fits
$ astimgwarp --matrix=0.2,0,0.4,0,0.2,0.4,0,0,1 image.fits
$ astimgwarp --matrix="0.7071,-0.7071  0.7071,0.7071" image.fits
```

ImageWarp can accept two arguments, one (the input image) is mandatory if any processing is to be done. An optional argument is a plain text file that will keep the warp/transform matrix, see Section 6.4.1 [Warping basics], page 110. There is also the `--matrix` option from which the matrix can be literally specified on the command-line. If both are present when calling ImageWarp, the contents of the plain text file have higher precedence. The general options to all Gnuastro programs can be seen in Section 4.1.4 [Common options], page 48.

By default the WCS (World Coordinate System) information of the input image is going to be corrected in the output image. WCSLIB will save the input WCS information in the `PC` matrix[34]. To correct the WCS, ImageWarp multiplies the `PC` matrix with the inverse of the specified transformation matrix. Also the `CRPIX` point is going to be changed to its correct place in the output image coordinates. This behavior can be disabled with the `--nowcscorrection` option.

To be the most accurate the input image will be converted to double precision floating points and all the processing will be done in this format. By default, in the end, the output image will be converted back to the input image data type. Note that if the input type was not a floating point format, then the floating point output pixels are going to be rounded to the nearest integer (using the `round` function in the C programming language) which can

[33] `http://en.wikipedia.org/wiki/Aliasing`

[34] Greisen E.W., Calabretta M.R. (2002) Representation of world coordinates in FITS. Astronomy and Astrophysics, 395, 1061-1075.

lead to a loss of data. This behavior can be disabled with the `--doubletype` option. The input file and input warping matrix elements are stored in the output's header.

Coordinates of pixel center: Based on the FITS standard, integer values are assigned to the center of a pixel and the coordinate [1.0, 1.0] is the center of the bottom left (first) image pixel. So the point [0.0, 0.0] is half a pixel away (in each axis) from the bottom left vertice of the first pixel[35].

`-m`
`--matrix` (`=STR`) The warp/transformation matrix. All the elements in this matrix must be separated by any number of space, tab or comma (`,`) characters. If you want to use the first two, then be sure to wrap the matrix within double quotation marks (`"`) so they are not confused with other arguments on the command-line, see Section 4.1.3 [Options], page 46. This also applies to values in the configuration files, see Section 4.2.1 [Configuration file format], page 52. The transformation matrix can be either 2 by 2 or 3 by 3 array, see Section 6.4.1 [Warping basics], page 110.

The determinant of the matrix has to be non-zero and it must not contain any non-number values (for example infinities or NaNs). The elements of the matrix have to be written row by row. So for the general homography matrix of Section 6.4.1 [Warping basics], page 110, it should be called with `--matrix=a,b,c,d,e,f,g,h,1`.

`--hstartwcs`
(`=INT`) Specify the first header keyword number (line) that should be used to read the WCS information, see the full explanation in Section 6.1.4 [Invoking ImageCrop], page 78.

`--hendwcs`
(`=INT`) Specify the last header keyword number (line) that should be used to read the WCS information, see the full explanation in Section 6.1.4 [Invoking ImageCrop], page 78.

`-n`
`--nowcscorrection`
Do not correct the WCS information of the input image and save it untouched to the output image.

`-z`
`--zerofornoinput`
Set output pixels which do not correspond to any input to zero. By default they are set to blank pixel values, see Section 6.1.3 [Blank pixels], page 77.

[35] So if you want to warp the image relative to the bottom left vertice of the bottom left pixel, you have to shift the warping center by [0.5, 0.5], apply your transform then shift back by [-0.5, -0.5]. Similar to the example in see Section 6.4.2 [Merging multiple warpings], page 112. For example see the one line example above which scales the image by one fifth (0.2). Without this correction (if it was `0.2,0,0,0,0.2,0,0,0,1`), the image would not be correctly scaled.

`-b`
`--maxblankfrac`
(`=FLT`) The maximum fractional area of blank pixels over the output pixel. If an output pixel is covered by blank pixels (see Section 6.1.3 [Blank pixels], page 77) for a larger fraction than the value to this option, the output pixel will be set to a blank pixel its self.

When the fraction is lower, the sum of non-blank pixel values over that pixel will be multiplied by the inverse of this fraction to correct for its flux and not cause discontinuities on the edges of blank regions. Note that even with this correction, discontinuities (very low non-blank values touching blank regions in the output image) might arise depending on the transformation and the blank pixels. So if there are blank pixels in the image, a good value to this option has to be found for that particular image and warp.

`-d`
`--doubletype`
By default the output image is going to have the same type as the input image. If this option is called, the output will be in double precision floating point format irrespective of the input data type. When dealing with integer input formats, this option can be useful in checking the results.

6.5 SubtractSky

Raw astronomical images (and even poorly processed images) don't usually have a uniform 'sky' value over their surface prior to processing, see Section 6.5.1 [Sky value], page 116, for a complete definition of the sky value. However, a uniform sky value over the image is vital for further processing. For ground based images (particularly at longer wavelengths) this can be due to actual variations in the atmosphere. Another cause might be systematic biases in the instrument or prior processing. For example stray light in the telescope/camera or bad flat-fielding or bias subtraction. The latter is a major issue in space based images where the atmosphere is no longer a problem.

SubtractSky is a tool to find the sky value and its standard deviation on a grid over the image. The size of the grid will determine how accurately it can account for gradients in the sky value. Such that if the gradient (change of sky value) is too sharp, a smaller grid size has to be chosen. However the results will be most accurate with a larger grid size.

6.5.1 Sky value

The discussion here is taken from Akhlaghi and Ichikawa (2015)[36]. Let's assume that all instrument defects – bias, dark and flat – have been corrected and the brightness (see Section 8.1.3 [Flux Brightness and magnitude], page 167) of a detected object, O, is desired. The sources of flux on pixel i[37] of the image can be written as follows:

- Contribution from the target object, (O_i).
- Contribution from other detected objects, (D_i).

[36] See the section on sky in Akhlaghi M., Ichikawa. T. (2015). Astrophysical Journal Supplement Series.

[37] For this analysis the dimension of the data (image) is irrelevant. So if the data is an image (2D) with width of w pixels, then a pixel located on column x and row y (where all counting starts from zero and (0, 0) is located on the bottom left corner of the image), would have an index: $i = x + y \times w$.

- Undetected objects or the fainter undetected regions of bright objects, (U_i).
- A cosmic ray, (C_i).
- The background flux, which is defined to be the count if none of the others exists on that pixel, (B_i).

The total flux in this pixel, T_i, can thus be written as:

$$T_i = B_i + D_i + U_i + C_i + O_i.$$

By definition, D_i is detected and it can be assumed that it is correctly subtracted, so that D_i can be set to zero. There are also methods to detect and remove cosmic rays (for example, van Dokkum (2001)[38]) enabling us to set $C_i = 0$. Note that in practice, D_i and U_i are correlated, because they both directly depend on the detection algorithm and its input parameters. Also note that no detection or cosmic ray removal algorithm is perfect. With these limitations in mind, the observed sky value for this pixel (S_i) can be defined as

$$S_i = B_i + U_i.$$

Therefore, as the detection process (algorithm and input parameters) becomes more accurate, or $U_i \to 0$, the sky value will tend to the background value or $S_i \to B_i$. Therefore, while B_i is an inherent property of the data (pixel in an image), S_i depends on the detection process. Over a group of pixels, for example in an image or part of an image, this equation translates to the average of undetected pixels. With this definition of sky, the object flux in the data can be calculated with

$$T_i = S_i + O_i \quad \to \quad O_i = T_i - S_i.$$

Hence, the more accurately S_i is measured, the more accurately the brightness (sum of pixel values) of the target object can be measured (photometry). Similarly, any under-(over-)estimation in the sky will directly translate to an over(under) estimation of the measured object's brightness. In the fainter outskirts of an object a very small fraction of the photo-electrons in the pixels actually belong to objects. Therefore even a small over estimation of the sky value will result in the loss of a very large portion of most galaxies. Based on the definition above, the sky value is only correctly found when all the detected objects (D_i and C_i) have been removed from the data.

6.5.1.1 Finding the sky value

This technique to find the sky value in a distribution was initially proposed in Akhlaghi and Ichikawa 2015[39].

Note that through the difference of the mode and median we have actually 'detected' data in the distribution. However this detection was only based on the total distribution of

[38] van Dokkum, P. G. (2001). Publications of the Astronomical Society of the Pacific. 113, 1420.

[39] Akhlaghi M., Ichikawa. T. (2015). Astrophysical Journal Supplement Series.

the data, not its spatial position in each mesh. So we adhere to the definition of Sky value in Section 6.5.1 [Sky value], page 116. Finding the median is very easy, the main problem is in finding the mode through a robust method. In Appendix C of Akhlaghi and Ichikawa (2015) a new approach to finding the mode in any astronomically relevant distribution is introduced.

Even when the mode and median are approximately equal, Cosmic rays can significantly bias the calculation of the average. Even if they are very few. However, usually, Cosmic rays have very sharp boundaries and do not fade away into the noise. Therefore, when the histogram of the distribution is plotted, they are clearly separate from the rest of the data. For example see Figure 15 in Akhlaghi and Ichikawa (2015). In such cases, σ-clipping is a perfect tool to remove the effect of such objects in the average and standard deviation. See Section 7.1.2 [Sigma clipping], page 129, for a complete explanation. So after asserting that the mode and median are approximately equal in a mesh (see Section 6.5.2 [Tiling an image], page 119), convergence-based σ-clipping is also applied before getting the final sky value and its standard deviation for a mesh.

6.5.1.2 Sky value misconceptions

As defined in Section 6.5.1 [Sky value], page 116, the sky value is only accurately defined when the detection algorithm is not significantly reliant on the sky value. In particular its detection threshold. However, most signal-based detection tools[40] used the sky value as a reference to define the detection threshold. So these old techniques had to rely on approximations based on other assumptions about the data. A review of those other techniques can be seen in Appendix A of Akhlaghi and Ichikawa (2015)[41]. Since they were extensively used in astronomical data analysis for several decades, such approximations have given rise to a lot of misconceptions, ambiguities and disagreements about the sky value and how to measure it. As a summary, the major methods used until now were an approximation of the mode of the image pixel distribution and σ-clipping.

- To find the mode of a distribution those methods would either have to assume (or find) a certain probability density function (PDF) or use the histogram. But the image pixels can have any distribution, and the histogram results are very inaccurate (there is a large dispersion) and depend on bin-widths.
- Another approach was to iteratively clip the brightest pixels in the image (which is known as σ-clipping, since the reference was found from the image mean and its standard deviation or σ). See Section 7.1.2 [Sigma clipping], page 129, for a complete explanation. The problem with σ-clipping was that real astronomical objects have diffuse and faint wings that penetrate deeply into the noise. So only removing their brightest parts is completely useless in removing the systematic bias an object's fainter parts cause in the sky value.

As discussed in Section 6.5.1 [Sky value], page 116, the sky value can only be correctly defined as the average of undetected pixels. Therefore all such approaches that try to approximate the sky value prior to detection are ultimately poor approximations.

[40] According to Akhlaghi and Ichikawa (2015), signal-based detection is a detection process that relies heavily on assumptions about the to-be-detected objects. This method was the most heavily used technique prior to the introduction of NoiseChisel in that paper.

[41] Akhlaghi M., Ichikawa. T. (2015). Astrophysical Journal Supplement Series.

6.5.2 Tiling an image

Some of the programs in Gnuastro will need to divide the pixels in an image into individual tiles or a mesh grid to be able to deal with gradients. In this section we will explain the concept in detail and how the user can check the grid. In the case of SubtractSky, if the image is completely uniform then one sky value will suffice for the whole image. See Section 6.5.1 [Sky value], page 116, for the definition of the sky value. Unfortunately though, as discussed in Section 6.5 [SubtractSky], page 116, in most images taken with ground or space-based telescopes the sky value is not uniform. So we have to break the image up into small tiles on a mesh grid, assume the sky value is constant over them and find the sky value on those tiles.

Meshes are considered to be a square with a side of `--meshsize` pixels. The best mesh size is directly related to the gradient on the image. In practice we assume that the gradient is not significant over each mesh. So if there is a strong gradient (for example in long wavelength ground based images) or the image is of a crowded area where there isn't too much blank area, you have to choose a smaller mesh size. A larger mesh will give more pixels and so the scatter in the results will be less.

For raw image processing, a simple mesh grid is not sufficient. Raw images are the unprocessed outputs of the camera detectors. Large detectors usually have multiple readout channels each with its own amplifier. For example the Hubble Space Telescope Advanced Camera for Surveys (ACS) has four amplifiers over its full detector area dividing the square field of view to four smaller squares. Ground based image detectors are not exempt, for example each CCD of Subaru Telescope's Hyper Suprime-Cam camera (which has 104 CCDs) has four amplifiers, but they have the same height of the CCD and divide the width by four parts.

The bias current on each amplifier is different, and normally bias subtraction is not accurately done. So even after subtracting the measured bias current, you can usually still identify the boundaries of different amplifiers by eye. See Figure 11(a) in Akhlaghi and Ichikawa (2015) for an example. This results in the final reduced data to have non-uniform amplifier-shaped regions with higher or lower background flux values. Such systematic biases will then propagate to all subsequent measurements we do on the data (for example photometry and subsequent stellar mass and star formation rate measurements in the case of galaxies). Therefore an accurate sky subtraction routine should also be able to account for such biases.

To get an accurate result, the mesh boundaries should be located exactly on the amplifier boundaries. Otherwise, some meshes will contain pixels that have been read from two or four different amplifiers. These meshes are going to give very biased results and the amplifier boundary will still be present after sky subtraction. So we define 'channel's. A channel is an independent mesh grid that covers one amplifier to ensure that the meshes do not pass the amplifier boundary. They can also be used in subsequent steps as the area used to identify nearby neighbors to interpolate and smooth the final grid, see Section 6.5.2.2 [Grid interpolation and smoothing], page 121. The number of channels along each axis can be specified by the user at run time through the command-line `--nch1` and `--nch2` options or in the configuration files, see Section 4.2 [Configuration files], page 51. The area of each channel will then be tiled by meshes of the given size and subsequent processing will be done on those meshes. If the image is processed or the detector only has one amplifier, you can set the number of channels in both axises to 1.

Unlike the channel size, that has to be an exact multiple of the image size, the mesh size can be any number. If it is not an exact multiple of the image side, the last mesh (rightest, for the first FITS dimension, and highest for the second when viewed in SAO ds9) will have a different size than the rest. If the remainder of the image size divided by mesh size is larger than a certain fraction (value to `--lastmeshfrac`) of the mesh size along each axis, a new (smaller) mesh will be put there instead of a larger mesh. This is done to avoid the last mesh becoming too large compared to the other meshes in the grid. Generally, it is best practice to choose the mesh size such that the last mesh is only a few (negligible) pixels wider or thinner than the rest.

The final mesh grid can be seen on the image with the `--checkmesh` option that is available to all programs which use the mesh grid for localized operations. When this option is called, a multi-extension FITS file with a `_mesh.fits` suffix will be created along with the outputs, see Section 4.5 [Automatic output], page 57. The first extension will be the input image. For each mesh grid the image produces, there will be a subsequent extension. Each pixel in the grid extensions is labeled to the mesh that it is part of. You can flip through the extensions to check the mesh sizes and locations compared to the input image.

6.5.2.1 Quantifying signal in a mesh

Noise is characterized with a fixed background value and a certain distribution. For example, for the Gaussian distribution these two are the mean and standard deviation. When we have absolutely no signal and only noise in a data set, the mean, median and mode of the distribution are equal within statistical errors and approximately equal to the background value. For the next paragraph, let's assume that the background is subtracted and is zero.

Data always has a positive value and will never become negative, see Figure 1 in Akhlaghi and Ichikawa (2015). Therefore, as data is buried into the noise, the mean, median and mode shift to the positive. The mean is the fastest in this shift. The median is slower since it is defined based on an ordered distribution and so is not affected by a small (less than half) number of outliers. Finally, the mode is the slowest to shift to the positive.

Inverting the argument above provides us with the basis of Gnuastro's algorithm to quantify the presence of signal in a mesh. Namely, when the mode and median of a distribution are approximately equal, we can argue that there is no significant signal in that mesh. So we can consider the image to be made of a grid and use this argument to 'detect' signal in each grid element. The median is defined to be the value of the 0.5 quantile in the image. So the only necessary parameter is the minimum acceptable quantile (smaller than 0.5) for the mode in a mesh that we deem accurate. See Section 6.5.2.4 [Mesh grid options], page 123, for an explanation of the options used to customize this behavior.

Since there is sufficient signal in the mesh to bias the analysis on that mesh, any grid element whose mode quantile is smaller than the minimum acceptable quantile is usually kept with a blank value and no value is given to it. Finally, when all the grid elements have been checked, we can interpolate over all the empty elements and smooth the final result to find the sky value over the full image. See Section 6.5.2.2 [Grid interpolation and smoothing], page 121.

Convolving a data set (that contains signal and noise), creates a positive skewness in it depending on the fraction of data present in the distribution and also the convolution kernel. See Section 3.1.1 in Akhlaghi and Ichikawa (2015) and Section 6.3.1.1 [Convolution

process], page 89. This skewness can be interpreted as an increase in the Signal to noise ratio of the objects buried in the noise. Therefore, to obtain an even better measure of the presence of signal in a mesh, the image can be convolved with a given PSF first. This positive skew will result in more distance between the mode an median thereby enabling a more accurate detection of fainter signal, for example the faint wings of bright stars and galaxies. When convolving over a mesh grid, the pixels in each channel will be treated independently. This can be disabled with the `--fullconvolution` option. See Section 6.3.4 [Convolution kernel], page 106, for the respective options.

6.5.2.2 Grid interpolation and smoothing

On some of the grid elements, the desired value will not be found, for example the Sky value in Section 6.5.1.1 [Finding the sky value], page 117. This can happen for lots of reasons depending on the job that was to be done on a mesh, for example when a large galaxy is present in the image, no Sky value will be found for the grid elements that lie over it. However, the Sky value should be 'guessed' over that part of the image. We cannot just ignore those regions! To fill in such blank grid elements, we use interpolation.

Parametric interpolations like bi-linear, bicubic or spline interpolations are not used because they fail terribly on the edges of the image. For example see Figure 16 in Akhlaghi and Ichikawa (2015). They are also prone to cause significant gradients in the image. So to find the interpolated value for each grid element, Gnuastro will look at a certain number of the nearest neighbors. The exact number can be specified through `--numnearest`. The median value of those grid elements will be taken as the final value for each mesh. The median is chosen because on the fainter wings of bright objects, the mean can easily become biased. If the number of meshes with a good value is less than the value given to `--numnearest`, then the program will abort and notify you. In such cases you can either decrease the value to this option or set less restrictive requirements (for example a smaller `--minmodeq`, or larger/smaller meshs) at the expense of less accurate results.

By default the process above will occur on all the grid elements, not just the ones that are blank. This is done to avoid biased results on the faint wings of bright galaxies and stars (the PSF). Because their flux penetrates into the noise very slowly, it might not be possible to completely identify that flux and ignore that mesh. Therefore, if interpolation is only done on blank pixels, such false positives can cause a bias in the vicinity of bright objects, particularly after smoothing in the next step. In order to only interpolate over blank meshs and leave the values of the successful meshs untouched, the `--interponlyblank` option can be used.

Once all the grid elements are filled, the values given to all the meshs should be smoothed. This is because the median was used in the interpolation. The median is robust in the face of outliers, but there might be strong differences from one grid element to the next. By smoothing the grid, the variation between grid element to grid element will be far less. To smooth the mesh values, Gnuastro uses an average filter. An average filter is just a convolution of the mesh grid (spatial convolution with edge correction) by a kernel with all elements having an equal weight, see Section 6.3.1.1 [Convolution process], page 89. The kernel is a square. The length of the kernel edge can be set in units of meshs through the `--smoothwidth` option (which has to be an odd number as with any kernel). If a width of 1 is given for the kernel width, then no smoothing will take place.

By default the interpolation and smoothing explained above are done independently on each channel. In some circumstances, it might be preferable to do either one of these two steps on the whole image, independent of the channels. For example when there are gradients over the image and their variation over the image is stronger than the variation caused by the channels. Through the two options `--fullinterpolation` and `--fullsmooth` you can ask for interpolation or smoothing to use all the meshs in the image, not just those in the same channel of a mesh. Note that it is still very important that no mesh contain pixels from two channels. Since the pixels have been assigned to a mesh prior to these steps (see Section 6.5.2 [Tiling an image], page 119), there is no problem in this regard.

Even after smoothing, a simple visual examination of the values given to the pixels in each mesh over the image will give a very 'boxy' or pixelated impression. To our eyes it feels that it would be much better if the values could be smoothed in sub-mesh (pixel) scales to obtain a smooth variation. An example is the results of bi-linear and bi-cubic interpolations, see Figure 16 in Akhlaghi and Ichikawa (2015) for several examples. The reasons we are not doing that level of smoothing are two fold:

- The difference in value between neighboring meshs is not statistically significant. After smoothing, the variation between the neighboring mesh values will be very small. It does visually appear to be a lot when viewing in image viewers like SAO ds9. This is because such viewers use the minimum and maximum range of all the pixel values as a reference to choose the color given to each pixel. However compared to the standard deviation of the noise in the image, the different values of each mesh are completely negligible. Please confirm this for your self on your own datasets to clearly understand.
- Such pixel-based (not mesh-based) interpolation will be very time consuming. Since the difference between the neighboring meshs is statistically negligible, it is simply not worth the time investment.

6.5.2.3 Checking grid values

Programs that use the mesh grid to calculate some value, also have checking options (that start with `--check`). When such options are called, the program will create a specific output file depending on what you want to check. When the operation is done on the meshes, this file shows the values assigned to each mesh grid element. It has three extensions:

1. The value that was initially calculated for each grid element. If a mesh was not successful in providing a value, a NaN value is stored for that mesh.
2. The interpolated values, see Section 6.5.2.2 [Grid interpolation and smoothing], page 121.
3. The smoothed values, see Section 6.5.2.2 [Grid interpolation and smoothing], page 121.

If more than one value is calculated for each mesh (for example the mean and its standard deviation), then each step has that many extensions. By default, these check image outputs have the same size (pixels) as the input image. So all the pixels within a mesh are given the same value. This is useful if you want to later apply these values to the image through another program for example. There is a one to one correspondence between the input image pixels and the grid showing you the mesh values for that pixel.

In other situations, the input pixel resolution might not be important for you and you just want to see the relative mesh values over the image. In such cases, you can call the `--meshbasedcheck` option so the check image only has one pixel for each mesh. This image

will only be as big as the full mesh grid and there will be no world coordinate system. When the input images are really large, this can make a difference in both the processing of the programs and in viewing the images.

Another case when only one pixel for each mesh will be useful is when you might want to display the mesh values in a document and you don't want the volume of the document (in bytes) to get too high. For example if the mesh sizes are 30 pixels by 30 pixels, then the mesh grid created through this option will take $30 \times 30 = 900$ times less space! You can resize the standard output, but the borders between the meshs will be blurred. The best case is to call that program with this option.

6.5.2.4 Mesh grid options

The mesh grid structure defined here (see Section 6.5.2 [Tiling an image], page 119) is used by more than one program. Therefore in order to avoid repetition, all the options to do with the mesh grid (and are shared by all the programs using it) are listed here.

Some programs might define multiple meshs over the image (for example in NoiseChisel, there is a large and a small mesh for different operations, see Section 7.2 [NoiseChisel], page 136), in such cases, the options for each mesh are designated by an appropriate prefix. For example in NoiseChisel the small and large mesh sizes are specified through the `--smeshsize` and `--lmeshsize` respectively. Both these options are similar to the `--meshsize` option explained below, but for their respective grid. Note that the short option name might also differ. If such options exist in a program, they are listed in the 'Invoking ProgramName' section within the list of options.

`-s`
`--meshsize`
: (`=INT`) The size of each mesh, see Section 6.5.2 [Tiling an image], page 119. If the width of all channels are not an exact multiple of the specified size, then the last mesh on each axis will have a different size to cover the full channel.

`-a`
`--nch1`
: (`=INT`) The number of channels along the first FITS axis (horizontal when viewed in SAO ds9). If the length of the image is not an exact multiple of this number, then the program will stop. You can use ImageCrop (Section 6.1 [ImageCrop], page 75) to trim off or add some pixels (blank pixels if added, see Section 6.1.3 [Blank pixels], page 77) to the image if it is not an exact multiple.

`-b`
`--nch2`
: (`=INT`) The number of channels along the second FITS axis, (vertical when viewed in SAO ds9). Similar to `--nch1`.

`-L`
`--lastmeshfrac`
: (`=FLT`) Fraction of extra area on the last (rightest on the first FITS axis and highest/top on the second) mesh, to define a new (smaller) one. See Section 6.5.2 [Tiling an image], page 119.

`--checkmesh`
: An image with suffix `_mesh.fits` will be created for you to check the mesh grid, see Section 6.5.2 [Tiling an image], page 119. The input image will be the

first extension, followed by an extension, where each pixel is labeled (number starting from zero) by the ID of the mesh it belongs to. If the program uses multiple mesh grids, the output will have more than two extensions. By flipping through the extensions, you can check the positioning and size of the meshs.

`-d`
`--mirrordist`
(`=FLT`) The distance beyond the mirror point (in units of the error in the mirror point) to check for finding the mode in each mesh. This is part of the process to quantify the presence of signal in a mesh, see Section 6.5.2.1 [Quantifying signal in a mesh], page 120. See appendix C in Akhlaghi and Ichikawa (2015) for a complete explanation of the mode-finding algorithm. The value to this option is shown as α in that appendix.

`-Q`
`--minmodeq`
(`=FLT`) The minimum acceptable quantile for the mode of each mesh. The median is on the 0.5 quantile of the image and as long as we have positive signal (all astronomically relevant observations), the mode will be less than the median. The sky value is only found on meshes where the median and mode are approximately the same, see Section 6.5.1 [Sky value], page 116.

`--interponlyblank`
Only interpolate the blank pixels. By default, interpolation will happen on all the mesh grids, not just the blank ones. See Section 6.5.2.2 [Grid interpolation and smoothing], page 121.

`-n`
`--numnearest`
(`=INT`) The number of nearest grid elements with a successful sky value to use for interpolating over blank mesh elements (those that had a significant contribution of signal), see Section 6.5.2.2 [Grid interpolation and smoothing], page 121.

`--fullinterpolation`
Do interpolation irrespective of the channels in the image, see Section 6.5.2.2 [Grid interpolation and smoothing], page 121.

`-T`
`--smoothwidth`
(`=INT`) The width of the average filter kernel used to smooth the interpolated image in units of pixels. See Section 6.5.2.2 [Grid interpolation and smoothing], page 121. If this option is given a value of 1 (one), then no smoothing will be done.

`--fullsmoothing`
Do smoothing irrespective of the channels in the image, see Section 6.5.2.2 [Grid interpolation and smoothing], page 121.

`--meshbasedcheck`
: Store the fixed value in each mesh in one check image pixel, see Section 6.5.2.3 [Checking grid values], page 122. Note that this image has no world coordinate system.

6.5.3 Mask image

All of the programs in Gnuastro that do processing on the input data can also account for masked pixels. Particularly in raw data processing, there are usually a set of pixels in the image that should not be included in any analysis. For example saturated pixels on the centers of bright objects, or the edges of the image which received no data and were only used for bias calculation. The detectors or the very early processing that is done on raw images clearly identifies such cases and usually assigns an integer (or flag or mask value) to those pixels. Pixels that are good for processing usually have a zero value in the mask image. It goes without saying that the two images have to have the same size. Note that if the mask image has blank pixels, then they act like pixels with non-zero values and will be masked (see Section 6.1.3 [Blank pixels], page 77).

The integer values in the mask images are usually sums of powers of two. Each power of two has a specific meaning and since the sum of two different sets of powers of two are never equal, each mask value identifies a set of different properties for each masked pixels. For some analysis, some masked properties might not be a problem. Therefore a pixel that only has that property should be included. In such cases, you can use bit flags to keep some of the powers of two and remove the rest. The individual Gnuastro programs do not consider these issues. Therefore, if some masked pixels should be included in the analysis, it is best to use another tool to set the appropriate mask pixel values to zero prior to running the analysis program. We are working on such a program as part of Gnuastro. If the data type (`BITPIX`) of the input mask image is not an integer type, the programs will print a warning, but continue on with the analysis. This usually happens because of a mistake in specifying the file or the HDU.

The programs that accept a mask image, all share the options below. Any masked pixels will receive a NaN value (or a blank pixel, see Section 6.1.3 [Blank pixels], page 77) in the final output of those programs. Infact, another way to notify any of the Gnuastro programs to not use a certain set of pixels in a data set is to set those pixels equal to appropriate blank pixel value for the type of the image, Section 6.1.3 [Blank pixels], page 77.

`-M`
`--mask` (`=STR`) Mask image file name. If this option is not given and the `--mhdu` option has a different value from `--hdu`, then the input image name will be used. If a name is specified on the command-line or in any of the configuration files, it will be used. If the program doesn't get any mask file name, it will use all the non-blank (see Section 6.1.3 [Blank pixels], page 77) pixels in the image. Therefore, specifying a mask file name in any of the configuration files is not mandatory.

`-H`
`--mhdu` (`=STR`) The mask image header name or number. Similar to the `--hdu` option, see Section 4.1.4 [Common options], page 48. Like `--mask`, this option does not have to be included in the configuration file or the command-line. However, if it is present on either of them, it will be used.

6.5.4 Invoking SubtractSky

SubtractSky will find the sky value on a grid in an image and subtract it. The executable name is `astsubtractsky` with the following general template:

```
$ astsubtractsky [OPTION ...] Image
```

One line examples:

```
$ astsubtractsky image.fits
$ astsubtractsky --hdu=0 --mhdu=1 image.fits
$ astsubtractsky --nch1=2 --nch2=2 image.fits
$ astsubtractsky --mask=maskforimage.fits image.fits
```

The only required input to SubtractSky is the input data file that currently has to be only a 2D image. But in the future it might be useful to use it for 1D or 3D data too. Any pixels in the image with a blank value will be ignored in the analysis, see Section 6.1.3 [Blank pixels], page 77. Alternatively a mask can be specified which indicates pixels to not be used (see `--mask` and `--mhdu`). The common options to all Gnuastro programs can be seen in Section 4.1.4 [Common options], page 48, and input data formats are explained in Section 4.1.2 [Arguments], page 46.

SubtractSky uses a mesh grid to tile the image. This enables it to deal with possible gradients, see Section 6.5.2 [Tiling an image], page 119. The mesh grid options are common to all the programs using it, and are listed in Section 6.5.2.4 [Mesh grid options], page 123. In order to ignore some pixels during the analysis, you can specify a mask image, see Section 6.5.3 [Mask image], page 125, for an explanation and the relevant options. For a more accurate result, a kernel file name can be specified so the image is first convolved, see Section 6.5.2.1 [Quantifying signal in a mesh], page 120, see Section 6.3.4 [Convolution kernel], page 106, for the kernel related options. The `--kernel` option is not mandatory and if not specified anywhere prior to running, the original image will be used. Please run SubtractSky with the `--help` option to list all the recognized options, irrespective of which part of the book they are fully explained in.

`-u`
`--sigclipmultip`
: (`=FLT`) The multiple of the standard deviation to clip from the distribution in σ-clipping. This is necessary to remove the effect of cosmic rays, see Section 6.5.1 [Sky value], page 116, and Section 7.1.2 [Sigma clipping], page 129.

`-t`
`--sigcliptolerance`
: (`=FLT`) The tolerance of sigma clipping. If the fractional change in the standard deviation before and after σ-clipping is less than the value given to this option, σ-clipping will stop, see Section 7.1.2 [Sigma clipping], page 129.

`--checksky`
: A two extension FITS image ending with `_smooth.fits` will be created showing how the interpolated sky value is smoothed.

`--checkskystd`
: In the interpolation and sky checks above, include the sky standard deviation too. By default, only the sky value is shown in all the checks. However with

this option, an extension will be added showing how the standard deviation on each mesh is finally found too.

7 Image analysis

Astronomical images contain very valuable information, the tools in this section can help in extracting and quantifying that information. For example calculating image statistics, or finding the sky value or detecting objects within an image.

7.1 ImageStatistics

The distribution of pixel values in an image can give us valuable information about the image, for example if it is a positively skewed distribution, we can see that there is significant data in the image. If the distribution is roughly symmetric, we can tell that there is no significant data in the image.

On the other hand, in some measurements that we do on the image, we might need to know the certain statistical parameters of the image. For example, if we have run a detection algorithm on an image, and we want to see how accurate it was, one method is to calculate the average of the undetected pixels and see how reasonable it is (if detection is done correctly, the average of undetected pixels should be approximately equal to the background value, see Section 6.5.1 [Sky value], page 116). ImageStatistics is built for precisely such situations.

7.1.1 Histogram and Cumulative Frequency Plot

Histograms and the cumulative frequency plots are both used to study the distribution of data. The histogram is mainly easier to understand for the untrained eye, while the cumulative frequency plot is more accurate, but needs a good level of experience for interpretation.

A histogram shows the number of data points which lie within pre-defined intervals (bins). It is used to get a general view of the distribution and its shape. The width of the bins is one of the most important parameters for a histogram. In the limiting case that the bin-widths tend to zero (and assuming there is data for each bin), then the normalized histogram would show the probability distribution function of the distribution. Normalizing a histogram means to divide the number of data points in each bin by the total number of data.

In the cumulative frequency plot of a distribution, the x axis is the sorted data values and the y axis is the index of each data in the sorted distribution. Unlike a histogram, a cumulative frequency plot does not involve intervals or bins. This makes it less prone to any sort of bias or error that a given bin-width would have on the analysis. When a larger number of the data points have roughly the same value, then the cumulative frequency plot will become steep in that vicinity. This occurs because on the x axis (data values), there is little change while on the y axis the indexes constantly increase. Normalizing a cumulative frequency plot means to divide each index (y axis) by the total number of data points.

Unlike the histogram which has a limited number of bins, ideally the cumulative frequency plot should have one point for every data point. Even in small images (for example a 200×200) this will result in an unreasonably larger number of points to plot (40000)! So when the cumulative frequency plot of an image is stored in a text file, it is best to only store its value on a certain number of points (intervals) rather than the whole data. The number of points to use for the final plot can be specified with the `--cfpnum` option.

Note that the interval's lower value is considered to be part of each interval, but its larger value is not. Formally, an interval between a and b is represented by [a, b). This is true for all the intervals except the last one. The last interval is closed or [a, b].

7.1.2 Sigma clipping

Let's assume that you have pure noise (centered on zero) with a clear Gaussian distribution, see Section 8.2.1.1 [Photon counting noise], page 176. Now let's assume you add very bright objects (signal) on the image which have a very sharp boundary. By a sharp boundary, we mean that there is a clear cutoff at the place the objects finish. In other words, at their boundaries, the objects do not fade away into the noise. In such a case, when you plot the histogram (see Section 7.1.1 [Histogram and Cumulative Frequency Plot], page 128) of the distribution, the pixels relating to those objects will be clearly separate from pixels that belong to parts of the image that did not have data. In the cumulative frequency plot, you would observe a long flat region were for a certain range of data (x axis), there is no increase in the index (y axis).

In cases like the above, σ-clipping is a very useful tool to remove the effect (bias) of such forms of signal from the distribution. In astronomical applications, cosmic rays (when they collide at a near normal incidence angle) are a very good example of such signals. The tracks they leave behind in the image are perfectly immune to the blurring caused by the atmosphere and the aperture. They are also very energetic and so their borders are usually clearly separated from the surrounding noise. So σ-clipping is very useful in removing their effect on the data. See Figure 15 in Akhlaghi and Ichikawa (2015).

σ-clipping is the very simple iteration below. In each iteration, the range of input data might decrease and so when the data in the image have the conditions above, they will be removed. The criteria to stop the iteration will be discussed below.

1. Calculate the mean, standard deviation (σ) and median (m) of a distribution.
2. Remove all points that are smaller or larger than $m \pm \alpha\sigma$.

The reason the median is used as a reference and not the mean is that the mean is too significantly affected by the presence of signal, while the median is less affected, see Section 6.5.1.1 [Finding the sky value], page 117. As you can tell from this algorithm, besides the condition above (that the signal have clear high signal to noise boundaries) σ-clipping is only useful when the signal does not cover more than half of the full data set. If they do, then the median will lie over the signal and sigma clipping might remove the pixels with no signal.

In the literature researchers use either one of two criteria to stop this iteration above:

- When a certain number of iterations has taken place.
- When the new measured standard deviation is within a certain tolerance level of the old one. The tolerance level is defined by:

$$\frac{\sigma_{old} - \sigma_{new}}{\sigma_{new}}$$

  The standard deviation is heavily influenced by the presence of outliers. Therefore the fact that it stops changing between two iterations is a sign that we have successfully removed outliers. Note that in each clipping, the dispersion in the distribution is either less or equal. So $\sigma_{old} \geq \sigma_{new}$.

Other objects in astronomical data analysis like galaxies and stars are blurred by the atmosphere and the telescope aperture. Galaxies in particular do not appear to have a clear high signal to noise cutoff at all. Therefore σ-clipping will not be useful in removing their effect on the data. In astronomy, it is only useful for removing the effect of Cosmic rays.

7.1.3 Mirror distribution

The mirror distribution of a data set was defined in Appendix C of Akhlaghi and Ichikawa (2015). It is best visualized by mentally placing a mirror on the histogram of a distribution at any point within the distribution (which we call the mirror point).

Through the `--mirrorquant` in ImageStatistics, you can check the mirror of a distribution when the mirror is placed on any given quantile. The mirror distribution is plotted along with the input distribution both as histograms and cumulative frequency plots, see Section 7.1.1 [Histogram and Cumulative Frequency Plot], page 128. Unlike the rest of the histograms and cumulative frequency plots in ImageStatistics, the text files created with the `--mirrorquant` and `--checkmode` will contain 3 columns. The first is the horizontal axis similar to all other histograms and cumulative frequency plots. The second column shows the respective value for the actual data distribution and the third shows the value for the mirror distribution.

The value for each bin of both histogram is divided by the maximum of both. For the cumulative frequency plot, the value in each bin is divided by the maximum number of elements. So one of the cumulative frequency plots reaches the maximum vertical axis of 1. The outputs will have the `_mirrorhist.txt` and `_mirrorcfp.txt` suffixes respectively. You can use a simple Python script like the one below to display the histograms and cumulative frequency plots in one plot:

```
#! /usr/bin/env python3

# Import the necessary modules:
import sys
import numpy as np
import matplotlib.pyplot as plt

# Load the two files:
a=np.loadtxt(sys.argv[1]+"_mirrorhist.txt")
b=np.loadtxt(sys.argv[1]+"_mirrorcfp.txt")

# Calculate the bin width:
w=a[1,0]-a[0,0]

# Plot the two histograms and cumulative frequency plots:
plt.bar(a[:,0], a[:,1], width=w, color="blue", linewidth=0, alpha=0.6)
plt.bar(a[:,0], a[:,2], width=w, color="green", linewidth=0, alpha=0.4)
plt.plot(b[:,0], b[:,1], linewidth=2, color="blue")
plt.plot(b[:,0], b[:,2], linewidth=2, color="green")

# Write the axis labels:
plt.ylim([0,np.amax(a[:,1])])
```

```
plt.xlim([np.amin(a[:,0]),np.amax(a[:,0])])

# Save the output to any name from the command-line:
plt.savefig(sys.argv[1]+"_plot.pdf")
```

The output format can be anything that Python's Matplotlib recognizes (for example png or jpg are also acceptable). So, if your input data file name was `input.fits`, and you want to see how its mirror distribution would look like if the mirror was placed at the 0.8 quantile (or the value which is above 80 percent of your data) you can run the following sequence of commands to see the combined cumulative frequency plot and histogram together in one PDF file. Let's assume you have put the Python script above into the file `mirrorplot.py`. The second command makes the Python script an executable file.

```
$ ls
input.fits mirrorplot.py
$ chmod +x mirrorplot.py
$ astimgstat input.fits --nohist --nocfp --mirrorquant=0.8
$ ls
input.fits  input_mirrorcfp.txt  input_mirrorhist.txt  mirrorplot.py
$ ./mirrorplot.py input
```

By default, the range of the mirror distribution is set to the 0.01 quantile of the image to 2 times the distance of the mode to that point. This is done so that the mode (which will have a value of zero in this plot) is positioned exactly on 1/3rd point of the x axis plot. The number of bins is the same value as the `--histnumbins` option. Alternatively, through the option `--histrangeformirror`, the histogram properties set with the `--histmin` and `--histmax` can be used. In these mirror plots, the mirror value is going to have the value of zero and one of the bins is going to start at zero (if `--histrangeformirror` is called, the given ranges will also shift accordingly).

7.1.4 Invoking ImageStatistics

ImageStatistics will print the major statistical measures of an image's pixel value distribution. The executable name is `astimgstat` with the following general template

```
$ astimgstat [OPTION ...] InputImage.fits
```

One line examples:

```
$ astimgstat input.fits
$ astimgstat animage.fits --ignoremin --nohist
$ astimgstat anotherimage.fits --mask=detectionlabels.fits --mhdu=1
```

If ImageStatistics is to do any data processing, an input image should be provided with the recognized extensions as input data, see Section 4.1.2 [Arguments], page 46. See Section 4.1.4 [Common options], page 48, for the list of options that are shared by all programs. All the main statistical operations have their specific set of options. If a string is given to the `--output` option, it is used as the base name for the generated files. Without this option, the input image name is used as the name-base.

Some of the options are necessary and if they are not included in the configuration file, ImageStatistics will not run, see Section 4.2 [Configuration files], page 51. However, for some others this is not so: `--histmin`, `--histmax`, `--histquant`, `--cfpmin`, `--cfpmax`,

`--cfpquant`. These are options to do with the range of values in the histogram and cumulative frequency plots. If no value is given for these options when ImageStatistics is about to start processing the data, then the full data range will be used. Such that the minimum image value will be set for the minimums and the maximum image value will be used for the maximum. The Mask name and HDU are also not mandatory in the configuration file.

By default, in verbose mode[1], along with a short summary of the basic data statistics, a simple ASCII histogram will also be printed. This can be useful for a very quick and general view of the distribution. An example verbose output of ImageStatistics on one of the `$ make check` outputs can be seen below:

```
$ astimgstat ./tests/convolve_spatial_warped_noised.fits        \
             --histquant=0.05
ImageStatistics started on AAA BBB CC DD:EE:FF GGGG
  - Input read: ./tests/convolve_spatial_warped_noised.fits (hdu: 0)
   -- Number of points                                  10000
   -- Minimum                                           -38.2066
   -- Maximum                                           1268.72
   -- Sum                                               154927
   -- Mean                                              15.4927
   -- Standard deviation                                60.5407
   -- Median                                            4.82691
   -- Mode (quantile, value)                            0.4335, 2.90301
   -- Mode symmetricity and its cutoff value            0.5909, 17.2507
   -- ASCII histogram in the range: -13.912957  --  65.058487:
    |           * *
    |         ********
    |         ********* **
    |       **************
    |     ******************
    |    ********************
    |   **********************
    |  ************************
    |****************************
    |******************************
    |****************************************
    |************************************************************
    |------------------------------------------------------------


  - Sigma clipping results (Median, Mean, STD, Number):
    - 4.00 times sigma by convergence (tolerance: 0.2000):
       1: 4.826907, 15.492665, 60.540707, 10000
       2: 4.665353, 10.117773, 27.793823, 9881
       3: 4.433090, 7.458610, 18.510950, 9715
       4: 4.216213, 6.176818, 15.231371, 9575
       5: 4.088199, 5.679162, 14.183748, 9502
    - 4.00 sigma-clipping 5 times:
```

[1] If the `-q` option is not called then all programs will operate in verbose mode, see Section 4.1.4 [Common options], page 48.

```
 1: 4.826907, 15.492665, 60.540707, 10000
 2: 4.665353, 10.117773, 27.793823, 9881
 3: 4.433090, 7.458610, 18.510950, 9715
 4: 4.216213, 6.176818, 15.231371, 9575
 5: 4.088199, 5.679162, 14.183748, 9502
ImageStatistics finished in:  0.006964 (seconds)
```

`-M`
`--mask` (`=STR`) The file name of a mask image. If this option is not given on the command-line or in the configuration files and `--mhdu` is not given or is identical to `--hdu`, then no mask image will be used.

`-H`
`--mhdu` (`=STR`) The header name of the mask extension.

`-r`
`--ignoremin`
Ignore all data elements that have a value equal to the minimum value in the image. In practice this is like masking those pixels, their values will not be used.

`-l`
`--lowerbin`
Set the first column of the histogram and cumulative frequency plots to the lower interval boundary. By default (without calling this option), the central interval value is used.

`--onebinvalue`
(`=FLT`) Make sure that one bin starts with the value to this option. In practice, this will shift the bins used to find the histogram and cumulative frequency plot such that one bin's lower interval becomes this value. For example when the histogram range includes negative values, but the data doesn't. If zero is somewhere between one bin, then the viewers of the plot(s) will think negative data is also present. By setting `--onebinvalue=0`, you can make sure that the viewers of the histogram will not be confused.

Note that by default, the first row of the histogram and cumulative frequency plot show the central values of each bin. So in the example above you will not see the 0.000. To see it, add the `--lowerbin` option to show the lower value of each bin. If you don't care about the bin positions within the specified range you can set the value to this option to a Not-a-Number (NaN) value on the command-line (`--onebinvalue=nan`) or in the configuration files with a `nan` following the option name. If the value is not within the specified bin range, it will be ignored.

`--noasciihist`
Do not show an ASCII plot on the command-line.

`--mirrorquant`
(`=FLT`) quantile to put the mirror. A value between 0 and 1. See Section 7.1.3 [Mirror distribution], page 130, for a complete explanation. Outputs two files with suffixes `_mirrorhist.txt` and `_mirrorcfp.txt`.

`--checkmode`
: The mode of the data is found by comparing the input data distribution with its mirror distribution. If this option is called, the mirror distribution's histogram and cumulative frequency plots will be saved in to plain text files ending with `_modehist.txt` and `_modecfp.txt`. See the explanation for Section 7.1.3 [Mirror distribution], page 130, for more details about these two files and how you can easily plot the outputs. This option only works when when ImageStatistics is in verbose mode. Since otherwise the mode is not calculated.

  To draw the plots you can use the script in Section 7.1.3 [Mirror distribution], page 130. Just change the appended suffixes in the two calls to `np.loadtxt` in the Python script.

`--histrangeformirror`
: Use `--histmin` and `--histmax` for the range of the mirror distributions (which are produced with the `--mirrorquant` and `--checkmode` options).

Histogram: The stored histogram is stored in a text file ending with `_hist.txt`.

`--nohist` Do not calculate or save the histogram.

`--normhist`
: Make a normalized histogram, see Section 7.1.1 [Histogram and Cumulative Frequency Plot], page 128.

`--maxhistone`
: Divide all histogram bins by the number in the bin with the most data points. This is very useful if you want to plot the histogram along with a normalized cumulative frequency plot in one plot. Note that if the histogram numbers are important in showing along with the cumulative frequency plot, you can use `--maxcfpeqmaxhist`, see below.

`-n`
`--histnumbins`
: (`=INT`) The number of bins in the histogram. Note that in practice, this is also equivalent to the number of rows in the output text file.

`-i`
`--histmin`
: (`=FLT`) The minimum value to use in the histogram. If `--histquant` is given, then any value given `--histmin` or `--histmax` is ignored.

`-x`
`--histmax`
: (`=FLT`) The maximum value to use in the histogram. Similar to `--histmin`.

`-Q`
`--histquant`
: (`=FLT`) Set the range of the histogram based on the image quantile. So `--histquant=0.05` is given, all the data from the 0.05 quantile to 0.95 quantile will be used in the histogram. This is useful when there is a small number of outliers in the image. Note that if this option is given, any (possible) value given to `--histmin` or `--histmax` are ignored.

Cumulative Frequency Plot: The cumulative frequency plot will be stored in a text file ending with `_cfp.txt`. To be more realistic, the average of the indexs in each interval is used as the second column, see Section 7.1.1 [Histogram and Cumulative Frequency Plot], page 128.

`--nocfp` Do not calculate or store the cumulative frequency plot.

`--normcfp`
Normalize the cumulative frequency plot, see Section 7.1.1 [Histogram and Cumulative Frequency Plot], page 128.

`--maxcfpeqmaxhist`
Set the maximum cumulative frequency plot value to the maximum value in the histogram (if it is to be created). This is a useful in plotting the histogram and cumulative frequency plots together when the histogram numbers are important.

`--cfpsimhist`
Set the range of the cumulative frequency plot and the number of points to store to the same range as the histogram. If the two are to be plotted together, this is very useful, since the first axis (column) of the two will become identical.

`-p`
`--cfpnum` (`=INT`) The number of points to store the cumulative frequency plot. They will be evenly distributed between the range of pixel values.

`-a`
`--cfpmin` (`=FLT`) The minimum value to use for the cumulative pixel value range. If `--cfpquant` is given, then any value given `--cfpmin` or `--cfpmax` is ignored.

`-n`
`--cfpmax` (`=FLT`) The maximum value to use for the cumulative pixel value range. Similar to `--cfpmin`.

`-U`
`--cfpquant`
(`=FLT`) Similar to `--histquant` but for the cumulative frequency plot.

σ-clipping: The result of each iteration of σ-clipping will be printed in the terminal for both types of sigma clipping: A certain number of times and convergence of the standard deviation.

`--nosigclip`
If this option is called, no σ-clipping will take place.

`-u`
`--sigclipmultip`
(`=FLT`) The multiple of the standard deviation above which to clip. This value is demonstrated by α in Section 7.1.2 [Sigma clipping], page 129.

`-t`
`--sigcliptolerance`
(`=FLT`) If the fractional difference of the standard deviation becomes less than this value, then σ-clipping will halt, see Section 7.1.2 [Sigma clipping], page 129.

`-g`
`--sigclipnum`
(`=INT`) The number of iterations for the case where the σ-clipping iteration stops after a certain number of runs.

7.2 NoiseChisel

Once raw data have gone through the initial reduction process (through the programs in Chapter 6 [Image manipulation], page 75). We are ready to derive scientific results out of them. Unfortunately in most cases, the scientifically interesting targets are deeply drowned in a sea of noise. NoiseChisel is Gnuastro's tool to detect signal in noise. Infact, NoiseChisel was the motivation behind creating Gnuastro and has a journal article devoted to its techniques: arXiv:1505.01664 (`http://arxiv.org/abs/1505.01664`), published in 2015 by the Astrophysical Journal Supplement Series 220.

Following the explanations for the options in Section 7.2.1.1 [NoiseChisel options], page 137, should provide a relatively good idea of how NoiseChisel works, but it is recommended to see the paper since it does a very thorough job at explaining the concepts and methods of NoiseChisel with abundant demonstrations for each step. However, the paper cannot undergo any further updates, but NoiseChisel will evolve: better algorithms or steps will be found, thus options will be added or removed. So this section is the definitive guide. Please follow the `NEWS` file with each release for such updates.

Detection is the process of separating signal from noise. In other words, after detection is complete, one set of data elements (pixels in an image) will be distinguished as signal and another set of the data elements will be noise. Segmentation is the process of labeling the detected pixels into possibly multiple components (objects). For example when two galaxies lie sufficiently close to each other to be detected as one object.

NoiseChisel was the first software to make use of a noise-based concept to detection and segmentation. In this method, instead of emphasizing on the signal and trying to guess the properties of the to-be-detected targets prior to detection (for example assuming that it is an ellipse), the emphasis is put on the noise in the image and it imposes statistically negligible requirements on the signal. The name of NoiseChisel is derived from the first thing it does after thresholding the image: to erode it. In mathematical morphology, erosion on pixels can be pictured as carving off boundary pixels. So what NoiseChisel does is similar to what a wood chisel or stone chisel do. It is just not a hardware but software.

7.2.1 Invoking NoiseChisel

NoiseChisel will detect signal in noise. The executable name is `astnoisechisel` with the following general template

```
$ astnoisechisel [OPTION ...] InputImage.fits
```

One line examples:

```
$ astnoisechisel input.fits
$ astnoisechisel --nch1=4 --lmesh=256 input.fits
$ astnoisechisel field.fits --mask=badpixels.fits --mhdu=1
```

If NoiseChisel is to do processing, an input image should be provided with the recognized extensions as input data, see Section 4.1.2 [Arguments], page 46. The options that are shared

by all Gnuastro programs can be seen in Section 4.1.4 [Common options], page 48. In order to ignore some pixels during the analysis, you can specify a mask image, see Section 6.5.3 [Mask image], page 125, for an explanation and the relevant options. A masked pixel is completely ignored.

A convolution kernel can also be optionally given. If a value (file name) is given to `--kernel` on the command-line or in a configuration file (see Section 4.2 [Configuration files], page 51), then that file will be used to convolve the image prior to thresholding. Otherwise a default kernel will be used. The default kernel is a 2D Gaussian with a FWHM of 2 pixels truncated at 5 times the FWHM. See Section 3.1.1 of Akhlaghi and Ichikawa (2015) to learn why this particular kernel was chosen as default. See Section 6.3.4 [Convolution kernel], page 106, for kernel related options.

NoiseChisel uses a mesh grid to tile the image. This enables it to deal with possible gradients, see Section 6.5.2 [Tiling an image], page 119. The mesh grid options are common to all the programs using it, and are listed in Section 6.5.2.4 [Mesh grid options], page 123. In particular, NoiseChisel uses two mesh grids: a large and a small one. The sizes of the meshs in each grid can be specified with the following two options (the `--meshsize` option is not recognized by NoiseChisel). The rest of the options explained in Section 6.5.2.4 [Mesh grid options], page 123, apply to both grids.

`-s`
`--smeshsize`
: Similar to `--meshsize` in Section 6.5.2.4 [Mesh grid options], page 123, but for the smaller mesh grid, which is most dependent on the gradients in the image.

`-l`
`--lmeshsize`
: Similar to `--meshsize` in Section 6.5.2.4 [Mesh grid options], page 123, but for the larger mesh grid, used for detection and segmentation Signal to noise ratio analysis.

Please run NoiseChisel with the `--help` option to list all the recognized options with a short explanation, irrespective of which part of the Gnuastro book they are fully explained in, see Section 4.6 [Getting help], page 57.

7.2.1.1 NoiseChisel options

The options particular to NoiseChisel are listed below. They are classified by context and also sorted in the same order that the operations are done on the image. See Akhlaghi and Ichikawa (2015) for a very complete, detailed and illustrated explanation of each step. Reading through the option explanations should be enough to obtain a general idea of how NoiseChisel works. Before the procedures explained by these options begin, the image is convolved with a kernel. The first group of options listed below are those that apply to both the detection and the segmentation processes.

`-E`
`--skysubtracted`
: If this option is called, it is assumed that the image has already been sky subtracted once. Knowing if the sky has already been subtracted once or not is very important in estimating the Signal to noise ratio of the detections and

clumps. In short an extra σ_{sky}^2 must be added in the error (noise or denominator in the Signal to noise ratio) for every flux value that is present in the calculation. This can be interpreted as the error in measuring that sky value when it was subtracted by any other program. See Section 3.3 in Akhlaghi and Ichikawa (2015) for a complete explanation.

`-B`
`--minbfrac`
(`=FLT`) Minimum fraction (value between 0 and 1) of blank (undetected) area in a mesh for it to be considered in measuring the following properties.

- Measuring the Signal to noise ratio of false detections during the false detection removal.
- Measuring the sky value (average of undetected pixels). Both before the removal of false detections and after it.
- Clump Signal to noise ratio in the blank regions.

Because of the PSF, astronomical objects, other than cosmic rays, never have a clear cutoff and commonly sink into the noise very slowly. Even below the very low thresholds used by NoiseChisel. So when a large fraction of the area of one mesh is covered by detections, it is very probable that their faint wings are present in the undetected regions. Therefore, to get an accurate measurement of the above parameters over the full mesh grid, meshs that harbor too many detected regions should be excluded.

`-F`
`--minnumfalse`
(`=INT`) The minimum number of 'psudo-detections' (in identifying false detections) or clumps (in identifying false clumps) in each large mesh grid. If their number is less than this value, this mesh will be left blank and filled during mesh interpolation, see Section 6.5.2.2 [Grid interpolation and smoothing], page 121.

The Signal to noise ratio of false detections and clumps in each mesh is found using the quantile of the Signal to noise ratio distribution of the psudo-detections and clumps over the undetected pixels in each mesh. If the number of Signal to noise ratio measurements in each mesh is not enough, the quantile will not be accurate. For example if you set `--detquant=0.99` (or the top 1 percent), then it is best to have at least 100 Signal to noise ratio measurements.

Detection is the process of separating the pixels in the image into two groups: 1) Signal and 2) Noise. Through the parameters below, you can customize the detection process in NoiseChisel.

`-t`
`--qthresh`
(`=FLT`) The quantile threshold to apply to the convolved image. The detection process begins with applying a quantile threshold to each of the small mesh grid elements, see Section 6.5.2 [Tiling an image], page 119. The quantile is only calculated for those meshs that don't have any significant signal within them, see Section 6.5.2.1 [Quantifying signal in a mesh], page 120.

The quantile value is a floating point value between 0 and 1. Assume that we have sorted the N data elements of a distribution (the pixels in each mesh on the convolved image). The quantile (q) of this distribution is the value of the element with an index of (the nearest integer to) $q \times N$ in the sorted data set. After thresholding is complete, we will have a binary (two valued) image. The pixels above the threshold are known as foreground pixels (have a value of 1) while those which lie below the threshold are known as background (have a value of 0).

`--checkthreshold`
Check the quantile threshold values on the mesh grid. A file suffixed with `_thresh.fits` will be created, see Section 6.5.2.3 [Checking grid values], page 122.

`-e`
`--erode` (`=INT`) The number of erosions to apply to the binary thresholded image. Erosion is simply the process of flipping (from 1 to 0) any of the foreground pixels that neighbor a background pixel. In a 2D image, there are two kinds of neighbors, 4-connected and 8-connected neighbors. You can specify which type of neighbors should be used for erosion with the `--erodengb` option, see below.

Erosion has the effect of shrinking the foreground pixels. To put it another way, it expands the holes. This is a founding principle in NoiseChisel: it exploits the fact that with very low thresholds, the holes in the very low surface brightness regions of an image will be smaller than regions that have no signal. Therefore by expanding those holes, we are able to separate the regions harboring signal.

`--erodengb`
(`=4or8`) The type of neighborhood (structuring element) used in erosion, see `--erode` for an explanation on erosion. In 4-connectivity, the neighbors of a pixel are defined as the four pixels on the top, bottom, right and left of a pixel that share an edge with it. The 8-connected neighbors on the other hand include the 4-connected neighbors along with the other 4 pixels that share a corner with this pixel. See Figure 6 (a) and (b) in Akhlaghi and Ichikawa (2015) for a demonstration.

`--noerodequant`
Pure erosion is going to carve off sharp and small objects completely out of the detected regions. This option can be used to avoid missing such sharp and small objects (which have significant pixels, but not over a large area). All pixels with a value larger than the significance level specified by this option will not be eroded during the erosion step above. However, they will undergo the erosion and dilation of the opening step below.

Like the `--qthresh` option, the significance level is determined using the quantile (a value between 0 and 1). Just as a reminder, in the normal distribution, 1σ, 1.5σ, and 2σ are approximately on the 0.84, 0.93, and 0.98 quantiles.

`-p`
`--opening`
(`=INT`) Depth of opening to be applied to the eroded binary image. Opening is a composite operation. When opening a binary image with a depth of n, n ero-

sions (explained in `--erode`) are followed by n dilations. Simply put, dilation is the inverse of erosion. When dilating an image any background pixel is flipped (from 0 to 1) to become a foreground pixel. Dilation has the effect of fattening the foreground. Note that in NoiseChisel, the erosion which is part of opening is independent of the initial erosion that is done on the thresholded image (explained in `--erode`). The structuring element for the opening can be specified with the `--openingngb` option. Opening has the effect of removing the thin foreground connections (mostly noise) between separate foreground 'islands' (detections) thereby completely isolating them. Once opening is complete, we have *initial* detections.

`--openingngb`
(`=4or8`) The structuring element used for opening, see `--erodengb` for more information about a structuring element.

`-u`
`--sigclumpmultip`
(`=FLT`) The multiple of the standard deviation during σ-clipping. NoiseChisel uses σ-clipping to remove the effect of cosmic rays when calculating the average and standard deviation of the undetected regions.

Since cosmic rays have sharp boundaries and are usually small, the erosion and opening might put them within the undetected pixels. Although they might cover a very small number of pixels, they usually have very large flux values which can easily bias the average and standard deviation measured on a mesh. Their effect can easily be removed by σ-clipping, see Section 7.1.2 [Sigma clipping], page 129. NoiseChisel uses the convergence of the value of the standard deviation as the criteria to stop the σ-clipping iteration.

`-r`
`--sigcliptolerance`
(`=FLT`) The tolerance level to stop σ-clipping. The iteration is stopped when $(\sigma_{old} - \sigma_{new})/\sigma_{new}$ becomes smaller than the value given to this option. Note that σ_{old} will always be larger than σ_{new}. Only statistical scatter (error) can cause it to be smaller, in which case they can be considered to be approximately equal.

`--checkdetectionsky`
Check the initial approximation of the sky value and its standard deviation in a FITS file ending with `_detsky.fits`. See Section 6.5.2.3 [Checking grid values], page 122, for more information. If `--meshbasedcheck` is not called, then the first extension will be the the binary image with initial detections labeled one and background labeled zero. The mesh values will be in the subsequent extensions.

`-R`
`--dthresh`
(`=FLT`) The detection threshold: a multiple of the initial sky standard deviation added with the initial sky approximation. This flux threshold is applied to the initially undetected regions on the unconvolved image. The background pixels that are completely engulfed in a 4-connected foreground region are converted

to background (holes are filled) and one opening (depth of 1) is applied over both the initially detected and undetected regions. The Signal to noise ratio of the resulting 'psudo-detections' are used to identify true vs. false detections. See Section 3.1.5 and Figure 7 in Akhlaghi and Ichikawa (2015) for a very complete explanation.

`-i`
`--detsnminarea`
(`=INT`) The minimum area to calculate the Signal to noise ratio on the psudo-detections of both the initially detected and undetected regions. When the area in a psudo-detection is too small, the Signal to noise ratio measurements will not be accurate and their distribution will be heavily skewed to the positive. So it is best to ignore any psudo-detection that is smaller than this area. Use `--detsnhistnbins` to check if this value is reasonable or not.

`--detsnhistnbins`
(`=INT`) If not equal to zero, a histogram of the Signal to noise ratios of the psudo-detections will be stored in a text file ending with `_detsn.txt`. The number of bins in this histogram is specified by the value given to this option. This is good for inspecting the best possible value to `--detsnminarea`.

An empirical way to estimate the best `--detsnminarea` value for your data set is that the histogram have a sharp drop towards the higher S/Ns. In other words, when there is a prominent peak in the histogram and the last few bins have less than 10 (an arbitrary number, meaning very few!) pseudo-detections. When the minimum area is too large or too small, the slope of the histogram in the higher S/N bins will not be to sharp with the very high S/N bins having a large number of objects and the peak being less significant. A good minimum area will result in the last bins having very few pseudo-detections.

`-c`
`--detquant`
(`=FLT`) The quantile of the Signal to noise ratio distribution of the psudo-detections in each mesh to use for filling the large mesh grid. Note that this is only calculated for the large mesh grids that satisfy the minimum fraction of undetected pixels (value of `--minbfrac`) and minimum number of psudo-detections (value of `--minnumfalse`).

`-I`
`--dilate` (`=INT`) Number of times to dilate the final true detections. See the explanations in `--opening` for more information on dilation. The structuring element for this final dilation is fixed to an 8-connected neighborhood. This is because astronomical objects, except cosmic rays, never have a clear cutoff, so all the 8-pixels connected to the border pixels of a detection might harbor data.

`--checkdetection`
Every step of the detection process will be added as an extension to a file with the suffix `_det.fits`. Going through each would just be a repeat of the explanations above and also of those in Akhlaghi and Ichikawa (2015). The extension label should be sufficient to recognize which step you are observing. Viewing all the steps can be the best guide in choosing the best set of parameters. Note

that calling this function will significantly slow NoiseChisel. Normally the detection steps are done in parallel, but to show you each step individually, the parallel processing has to be halted and restarted multiple times. Below are some notes that might be useful in interpreting certain steps, beyond the paper.

- Going from the first "Labeled" extension (for the background pseudo-detections, which shows the labeled pseudo-detections) to the next extension ("For S/N"), with a relatively low `--dthresh` value, you will notice that a lot of the large area pseudo-detections are removed and not used in calculating the S/N threshold. The main reason for this is that they overlap with possible detections. You can check by going to the next extension and seeing how there are detections there. The filled holes have been covering initial detections. If you have many such cases, it might be a good sign that your `--dthresh` value is too low.

`--checksky`
: Check the final sky and its standard deviation values on the mesh grid. Similar to `--checkdetectionsky`.

`--checkmaskdet`
: Mask (set to NaN) the undetected pixels in one extension and the detected pixels in the next extension of a file ending with `_maskdet.fits`.

Segmentation is the process of possibly breaking up a detection into multiple objects and clumps. In deep surveys segmentation becomes particularly important since galaxies might fall along the same line of sight or they might be merging. It is thus very important to be able separate the pixels within a detection if it is necessary. After segmentation, such a detected region will get different labels.

In NoiseChisel, segmentation is done by first finding the 'true' clumps over a detection and then expanding those clumps to a certain flux limit. True clumps are found in a process very similar to the true detections explained above, see Akhlaghi and Ichikawa (2015) for more information. If the connections between the grown clumps are weaker than a given threshold, the grown clumps are considered to be separate objects. Otherwise, the are considered to be part of the same object.

`--detectonly`
: If this option is called, no segmentation will be done. The object labels extension in the output will simply be the detection (connected components) labels and the clumps image will be blank (see Section 7.2.1.2 [NoiseChisel output], page 144).

  This option can result in faster processing when only the noise properties of the image are desired for a catalog using another image's labels. A common case is when you want to measure colors or SEDs in several images. Let's say you have images in two colors: A and B. For simplicity also assume that they are exactly on the same position in the sky with the same pixel scale.

  You choose to set A as a reference, so you run the full NoiseChisel on A. Then you run NoiseChisel on B with this option since you only need the noise properties of B (for the signal to noise column in its catalog). You can then run MakeCatalog on A normally, see Section 7.3 [MakeCatalog], page 145. To run

MakeCatalog on B, you simply set the object and clump labels images to those that NoiseChisel produced for A, see Section 7.3.3 [Invoking MakeCatalog], page 151.

`-m`
`--segsnminarea`
(`=INT`) The minimum area which a clump in the undetected regions should have in order to be considered in the clump Signal to noise ratio measurement. If this size is set to a small value, the Signal to noise ratio of false clumps will not be accurately found. Note that this has to be larger than the value to `--detsnminarea`. Because the clumps are found on the convolved (smoothed) image while the psudo-detections are found on the input image. Use `--clumpsnhistnbins` to check if this value is reasonable or not.

`--clumpsnhistnbins`
(`=INT`) Similar to `--detsnhistnbins` (see there for more explanations), but for the distribution of the S/N of clumps over the undetected regions. The relevant parameter to check here is be `--segsnminarea`.

`-g`
`--segquant`
(`=FLT`) The quantile of the noise clump Signal to noise ratio distribution. This value is used to identify true clumps over the detected regions.

`-v`
`--keepmaxnearriver`
Keep a clump whose maximum flux is 8-connected to a river pixel. By default such clumps over detections are considered to be noise and are removed irrespective of their brightness (see Section 8.1.3 [Flux Brightness and magnitude], page 167). Over large profiles, that sink into the noise very slowly, noise can cause part of the profile (which was flat without noise) to become a very large and with a very high Signal to noise ratio. In such cases, the pixel with the maximum flux in the clump will be immediately touching a river pixel.

`-G`
`--gthresh`
(`=FLT`) Threshold (multiple of the sky standard deviation added with the sky) to stop growing true clumps. Once true clumps are found, they are set as the basis to segment the detected region. They are grown until the threshold specified by this option.

`--grownclumps`
In the output (see Section 7.2.1.2 [NoiseChisel output], page 144) store the grown clumps (or full detected region if only one clump was present in that detection). By default the original clumps are stored as the third extension of the output, if this option is called, it is replaced with the grown clump labels.

`-y`
`--minriverlength`
(`=INT`) The minimum length of a river between two grown clumps for it to be considered in Signal to noise ratio estimations. Similar to `--segsnminarea` and

`--detsnminarea`, if the length of the river is too short, the Signal to noise ratio can be noisy and unreliable. Any existing rivers shorter than this length will be considered as non-existent, independent of their Signal to noise ratio. Since the clumps are grown on the input image, this value should best be similar to the value of `--detsnminarea`. Recall that the clumps were defined on the convolved image so `--segsnminarea` was larger than `--detsnminarea`.

`-O`
`--objbordersn`
(`=FLT`) The maximum Signal to noise ratio of the rivers between two grown clumps in order to consider them as separate 'objects'. If the Signal to noise ratio of the river between two grown clumps is larger than this value, they are defined to be part of one 'object'. Note that the physical reality of these 'objects' can never be established with one image, or even multiple images from one broad-band filter. Any method we devise to define 'object's over a detected region is ultimately subjective.

Two very distant galaxies or satellites in one halo might lie in the same line of sight and be detected as clumps on one detection. On the other hand, the connection (through a spiral arm or tidal tail for example) between two parts of one galaxy might have such a low surface brightness that they are broken up into multiple detections or objects. Infact if you have noticed, exactly for this purpose, this is the only Signal to noise ratio that the user gives into NoiseChisel. The 'true' detections and clumps can be objectively identified from the noise characteristics of the image, so you don't have to give any hand input Signal to noise ratio.

`--checksegmentation`
A file with the suffix `_seg.fits` will be created. This file keeps all the relevant steps in finding true clumps and segmenting the detections in various extensions. Having read the paper or the steps above, the extension name should be enough to understand which step each extension is showing. Examining this file can be an excellent guide in choosing the best set of parameters. Note that calling this function will significantly slow NoiseChisel.

7.2.1.2 NoiseChisel output

The default name and directory of the outputs are explained in Section 4.5 [Automatic output], page 57. NoiseChisel's default output (when none of the options starting with `--check` or the `--output` option are called) is one file ending with `_labeled.fits`. This file has the extensions listed below:

1. A copy of the input image, a copy is placed here for the following reasons:
   - By flipping through the extensions, a user can check how accurate the detection and segmentation process was.
   - All the inputs to MakeCatalog (see Section 7.3 [MakeCatalog], page 145) are included in this one file which makes the running of MakeCatalog after NoiseChisel very easy.
   - All masked pixels given to NoiseChisel will have a value of NaN in this image. So

if you decide to run MakeCatalog after NoiseChisel, there is no more need to feed the mask image to MakeCatalog too.

2. The object labels. Each pixel in the input image is given a label in this extension, the labels start from one. The total number of labels is stored as the value to the `NOBJS` keyword in the header of this extension. It is also printed in verbose mode.
3. The clump labels. All the pixels in the input image that belong to a true clump are given a positive label in this extension. The detected regions that were not a clump are given a negative value to clearly identify the noise from the detections. The total number of clumps in this image is stored in the `NCLUMPS` keyword of this extension and printed in verbose output.

   If the `--grownclumps` option is called, or a value of `1` is given to it in any of the configuration files, then instead of the original clump regions, the grown clumps will be stored in this extension. Note that if there is only one clump (or no clumps) over a detected region, then the whole detected region is given a label of 1.
4. The final sky value on each pixel. See Section 6.5.1 [Sky value], page 116, for a complete explanation. See Section 6.5.2.2 [Grid interpolation and smoothing], page 121, for an explanation on the boxy appearance of this image.
5. Similar to the previous mesh but for the standard deviation on each pixel.

7.3 MakeCatalog

Detecting and segmenting signal over an image results in labeled images where each pixel is labeled with the ID (an integer) that is specified by the detector program. But this labeled image by its self can hardly be of any scientific use. The job of MakeCatalog is to combine the input image, the noise properties and the labels of pixels into a catalog (a text file table) which can easily be used for high-level scientific interpretations.

NoiseChisel (Gnuastro's signal detection tool, see Section 7.2 [NoiseChisel], page 136) does not produce any catalog of the detected objects by its self. Only a labeled FITS image is output, see Section 7.2.1.2 [NoiseChisel output], page 144. The output of NoiseChisel can be directly fed into MakeCatalog to generate the catalog. Some of the reasons for making the catalog in a separate program[2] are listed below:

- Complexity: Adding in a catalog functionality to the detector program will add several more steps (and options) to its processing that can equally well be done outside of it. This makes following the code harder for a curious reader and also potentially adds bugs.

  Another advantage of less complexity is when the parameter you want to measure over one profile is not provided by the developers of MakeCatalog. You can simply open this tiny little program and add your desired calculation easily. In Section 7.3.4 [Adding new columns to MakeCatalog], page 160, we have explained all the necessary steps to add a new column to MakeCatalog. However, if making a catalog was part of NoiseChisel, it would require a lot of energy to understand all the steps in order to add desired parameter.
- Low level nature of Gnuastro: Making catalogs is a separate process from labeling (detecting and segmenting) the pixels. A user might want to do certain operations on

[2] Most existing software that do object detection also output a catalog, so this is not a common practice.

the labeled regions before creating a catalog for them. Another user might want the properties of the same pixels in another image (possibly from another broadband filter) for measuring the colors or SEDs for example.

Here is an example of doing both: suppose you have images in various broad band filters at various resolutions and orientations. The image of one color will thus not lie exactly on another or even be in the same scale. However, it is imperative that the same pixels be used in measuring the colors of galaxies.

Therefore NoiseChisel can be run on the reference image and ImageWarp (Section 6.4 [ImageWarp], page 109) can be applied to the labeled images to find the pixels to use in the other image. Then MakeCatalog can generate the final catalog for both targets. It is currently customary to warp the images to the same pixel grid, however, this is very harmful for the data and creates correlated noise. It is much more accurate to do the transformations on the labeled image.

7.3.1 Quantifying data limits

Different datasets (images in the case of MakeCatalog) have different noise properties and different detection methods (or one method with a different set of parameters) will have different abilities to detect or measure certain kinds of objects in an image. Therefore it is very important to quantify our ability to detect and measure signal in noise. In this section we discuss these limits that are very important in any analysis.

In astronomy, it is common to use the magnitude (a unit-less scale) and physical units, see Section 8.1.3 [Flux Brightness and magnitude], page 167. Therefore all the measured discussed here are commonly defined in units of magnitudes.

Depth

As we make more observations on one region of the sky and add the images over each other, we are able to decrease the standard deviation of the noise in each pixel[3]. Qualitatively, this decrease manifests its self by making fainter (per pixel) parts of the objects in the image more visible. Quantitatively, it increases the Signal to noise ratio, since the signal is fixed but the noise is decreased. It is very important to have in mind that here, noise is defined per pixel (or in the units of our data measurement), not per object.

You can think of the noise as muddy water that is completely covering a flat ground[4] with some regions higher than the others[5] in it. In this analogy, height is brightness. Let's assume that in your first observation the muddy water has just been stirred and you can't see anything through it. As you wait and make more observations, the mud settles down and the *depth* of the transparent water increases as you wait. The summits of hills begin to appear. As the depth of clear water increases, the parts of the hills with lower heights (less surface brightness) can be seen more clearly.

The outputs of NoiseChisel include the Sky standard deviation (σ) on every group of pixels (a mesh) that were calculated from the undetected pixels in that mesh, see Section 6.5.2 [Tiling an image], page 119, and Section 7.2.1.2

[3] This is true for any noisy data, not just astronomical images

[4] The ground is the sky value in this analogy, see Section 6.5.1 [Sky value], page 116. Note that this analogy only holds for a flat sky value across the surface of the image or ground.

[5] The peaks are (parts of) astronomical objects in this analogy.

[NoiseChisel output], page 144. Let's take σ_m as the median σ over the successful meshes in the image (prior to interpolation or smoothing, see Section 6.5.2.2 [Grid interpolation and smoothing], page 121).

On different instruments pixels have different physical sizes (for example in micro-meters, or spatial angle over the sky), nevertheless, a pixel is our unit of data collection. In other words, while quantifying the noise, the physical or projected size of the pixels is irrelevant. We thus define the *depth* of each dataset (or image) as the magnitude of σ_m.

As an example, the XDF survey covers part of the sky that the Hubble space telescope has observed the most (for 85 orbits) and is consequently very small (~ 4 arcmin^2). On the other hand, the CANDELS survey, is one of the widest multi-color surveys covering several fields (about 720 arcmin^2) but its deepest fields have only 9 orbits observation. The depth of the XDF and CANDELS-deep surveys in the near infrared WFC3/F160W filter are respectively 34.40 and 32.45 magnitudes. In a single orbit image, this same field has a depth of 31.32. Recall that a larger magnitude corresponds to less brightness, see Section 8.1.3 [Flux Brightness and magnitude], page 167.

Upper limit magnitude

Due to the noisy nature of data, it is possible to get arbitrarily low values for a faint object's brightness (or arbitrarily high magnitudes). Given the scatter caused by the noise, such small values are meaningless: another similar depth observation will give a radically different value. This problem is most common when you use one image/filter to generate target labels (which specify which pixels belong to which object, see Section 7.2.1.2 [NoiseChisel output], page 144, and Section 7.3 [MakeCatalog], page 145) and another image/filter to generate a catalog for measuring colors.

The object might not be visible in the filter used for the latter image, or the image *depth* (see above) might be much shallower. So you will get unreasonbly faint magnitudes. For example when the depth of the image is 32 magnitudes, a measurement that gives a magnitude of 36 for a ~ 100 pixel object is clearly unreliable. In another similar depth image, we might measure a magnitude of 30 for it, and yet another might give 33. Furthermore, due to the noise scatter so close to the depth of the dataset, the total brightness might actually get measured as a negative value, so no magnitude can be defined (recall that a magnitude is a base-10 logarithm).

Using such unreliable measurements will directly affect our analysis, so we must not use them. However, all is not lost! Given our limited depth, there is one thing we can deduce about the object's magnitude: we can say that if something actually exists here (possibly burried deep under the noise), it must have a magnitude that is fainter than an *upper limit magnitude*. To find this upper limit magnitude, we place the object's footprint (segmentation map) over random parts of the image where there are no detections, so we only have pure (possibly correlated) noise and undetected objects. Doing this a large number of times will give us a distribution of brightness values. The standard deviation (σ) of that distribution can be used to quantify the upper limit magnitude.

Traditionally, faint/small object photometry was done using fixed circular apertures (for example with a diameter of N arcseconds). In this way, the upper limit was like the depth discussed above: one value for the whole image. But with the much more advanced hardware and software of today, we can make customized segmentation maps for each object. The number of pixels (are of the object) used directly affects the final distribution and thus magnitude. Also the image correlated noise might actually create certain patters, so the shape of the object can also affect the result. So in MakeCatalog, the upper limit magnitude is found for each object in the image separately. Not one value for the whole image.

Completeness limit

As the surface brightness of the objects decreases, the ability to detect them will also decrease. An important statistic is thus the fraction of objects of similar morphology and brightness that will be identified with our detection algorithm/parameters in the given image. This fraction is known as completeness. For brighter objects, completeness is 1: all bright objects that might exist over the imagde will be detected. However, as we go to lower surface brightness objects, we fail to detect some and gradually we are not able to detect anything any more. For a given profile, the magnitude where the completeness drops below a certain level usually above 90% is known as the completeness limit.

Another important parameter in measuring completeness is purity: the fraction of true detections to all true detections. In effect purity is the measure of contamination by false detections: the higher the purity, the lower the contamination. Completeness and purity are anti-correlated: if we can allow a large number of false detections (that we might be able to remove by other means), we can significantly increase theh completeness limit.

One traditional way to measure the completeness and purity of a given sample is by embedding mock profiles in regions of the image with no detection. However in such a study we must be really careful to choose model profiles as similar to the target of interest as possible.

7.3.2 Measuring elliptical parameters

One of the most important class of parameters that are important for astronomical image analysis is the shape or morphology of the objects. In the case of galaxies (with a very rich variety of morphologies) studying the morphology of the detected objects is its self a is very complicated research subject which is a very active field of research now. So in this section, we will just discuss how the most basic morphological parameters are estimated: the elliptical parameters for a set of labeled pixels (that will be studied together and define our object of interest). These are: the major axis, the minor axis and the position angle along with the central position of the profile. The derivations below follow the SExtractor manual derivations but include more explanations.

Let's begin with one dimension for simplicity: Assume we have a set of N values (brightness for example) B_i, each at position x_i. The simplest parameter we can define is the geometric center of the object (x_g) (ignoring the brightness values): $x_g = (\sum_i x_i)/N$. *Moments* are defined to incorporate both the value (brightness) and position of the data. The first moment can be written as:

$$\overline{x} = \frac{\sum_i B_i x_i}{\sum_i B_i}$$

This is essentially the weighted (by B_i) average position. The geometric center (x_g, defined above) is a special case of this with all $B_i = 1$. The second moment is essentially the variance of the distributions:

$$\overline{x^2} \equiv \frac{\sum_i (B_i x_i - \overline{x})^2}{\sum_i B_i} = \frac{\sum_i B_i^2 x_i^2}{\sum_i B_i} - 2\overline{x}\frac{\sum_i B_i x_i}{\sum_i B_i} + \overline{x}^2 = \frac{\sum_i B_i^2 x_i^2}{\sum_i B_i} - \overline{x}^2$$

The last step was done from the definition of $\overline{x}$. So the square root of $\overline{x^2}$ is the positional standard deviation (along the one-dimensional) of this particular brightness distribution (B_i). Crudely (or qualitatively), you can think of its square root as the distance (from $\overline{x}$) which contains a specific amount of the flux (depending on the B_i distribution). Similar to the first moment, the geometric second moment can be found by setting all $B_i = 1$. So while the first moment quantified the position of the brightness distribution, the second moment quantifies how that brightness is dispersed about the first moment. In other words, it quantifies how "sharp" the object's image is.

Before continuing to two dimensions and the derivation of the elliptical parameters we will pause for an implementation technicality. You can ignore this paragraph if you don't want to implement these concepts. The basic definition (first fraction for $\overline{x^2}$) can be used without any major problem. However, using this fraction requires two runs over the data: one run to specify $\overline{x}$ and one run for $\overline{x^2}$, this can be slow. However, using the last fraction above, we can estimate both the first and second moments in one run (since the $-\overline{x}^2$ term can easily be added later). Using the last form creates a major technical issue however: due to the floating point accuracy. Suppose the object is located between pixels 10000 and 10020. While the object pixels are only distributed over 20 pixels (with a standard deviation < 20), the mean has a value of ~ 10000. The $\sum_i B_i^2 x_i^2$ will go to very very large values while the individual pixel differences will be much smaller, this will lower the accuracy of our calculation due to the limited accuracy of floating point operations. Since the variance only depends on the distance of each point from the mean, and for a constant/arbitrary K, $\overline{x - K} = \overline{x} - K$, we can calculate the second order moment using:

$$\overline{x^2} = \frac{\sum_i B_i^2 (x_i - K)^2}{\sum_i B_i} - (\overline{x} - K)^2$$

The closer K is to $\overline{x}$, the better (the sums of squares will involve smaller numbers), as long as K is within the object limits (in the example above: $10000 \leq K \leq 10020$), the floating point error induced in our calculation will be negligible. For the most simplest implementation, in order to find the second moments (variance) in each dimension, MakeCatalog takes K to be the position of the first pixel that is assigned to the object under each dimension. Since K is arbitrary and an implementation/technical detail, we will ignore it for the remainder of this discussion.

In two dimensions, the mean and variances can be written as:

$$\overline{x} = \frac{\sum_i B_i x_i}{B_i}, \quad \overline{x^2} = \frac{\sum_i B_i^2 x_i^2}{\sum_i B_i} - \overline{x}^2$$

$$\overline{y} = \frac{\sum_i B_i y_i}{B_i}, \quad \overline{y^2} = \frac{\sum_i B_i^2 y_i^2}{\sum_i B_i} - \overline{y}^2$$

$$\overline{xy} = \frac{\sum_i B_i^2 x_i y_i}{\sum_i B_i} - \overline{x} \times \overline{y}$$

If an elliptical profile's major axis lies exactly along the x axis, then $\overline{x^2}$ will be directly proportional with the major axis, $\overline{y^2}$ with its minor axis and $\overline{xy} = 0$. However, in reality we are not that lucky and (assuming galaxies can be parametrized as an ellipse) the major axis of galaxies can be in any direction on the image (in fact this is one of the core principles behind weak-lensing by shear estimation). So the purpose of the remainder of this section is to define a strategy to measure the position angle and axis ratio of some randomly positioned ellipses in an image, using the raw second moments that we have calculated above in our image coordinates.

Let's assume we have rotated the galaxy by θ, then the new second order moments can be written as:

$$\overline{x_\theta^2} = \overline{x^2}\cos^2\theta + \overline{y^2}\sin^2\theta - 2\overline{xy}\cos\theta\sin\theta$$

$$\overline{y_\theta^2} = \overline{x^2}\sin^2\theta + \overline{y^2}\cos^2\theta + 2\overline{xy}\cos\theta\sin\theta$$

$$\overline{xy_\theta} = \overline{x^2}\cos\theta\sin\theta - \overline{y^2}\cos\theta\sin\theta + \overline{xy}(\cos^2\theta - \sin^2\theta)$$

We can now find the best θ (θ_0) such that the major axis lies along the x_θ axis. This can be found by setting:

$$\left.\frac{\partial\overline{x_\theta^2}}{\partial\theta}\right|_{\theta_0} = 0$$

Taking the derivative, we get:

$$2\cos\theta_0\sin\theta_0(\overline{y^2} - \overline{x^2}) + 2(\cos^2\theta_0 - \sin^2\theta_0)\overline{xy} = 0$$

When $\overline{x^2} \neq \overline{y^2}$, we can write:

$$\tan 2\theta_0 = 2\frac{\overline{xy}}{\overline{x^2} - \overline{y^2}}.$$

MakeCatalog uses the standard C math library's `atan2` function to estimate θ_0, which we define as the position angle of the ellipse. To recall, this is the angle of the major axis of the ellipse with the x axis. By definition, when the elliptical profile is rotated by θ_0, then $\overline{xy_{\theta_0}} = 0$, $\overline{x_{\theta_0}^2}$ will be the extent of the maximum variance and $\overline{y_{\theta_0}^2}$ the extent of the minimum variance (which are perpendicular for an ellipse). Replacing θ_0 in the equations above for $\overline{x_\theta}$ and $\overline{y_\theta}$, we can get the semi-major (A) and semi-minor (B) lengths:

$$A^2 \equiv \overline{x_{\theta_0}^2} = \frac{\overline{x^2} + \overline{y^2}}{2} + \sqrt{\left(\frac{\overline{x^2} - \overline{y^2}}{2}\right)^2 + \overline{xy}^2}$$

$$B^2 \equiv \overline{y_{\theta_0}^2} = \frac{\overline{x^2}+\overline{y^2}}{2} - \sqrt{\left(\frac{\overline{x^2}-\overline{y^2}}{2}\right)^2 + \overline{xy}^2}$$

So as a summary, it is important to remember that the units of A and B are in pixels (same as the standard deviation) and that they represent the spatial light distribution of the object in both image dimensions if the object could be approximated as an ellipse. When the object cannot be represented as an ellipse, this interpretation breaks down: $\overline{xy_{\theta_0}} \neq 0$ and $\overline{y_{\theta_0}^2}$ will not be the direction of minimum variance.

7.3.3 Invoking MakeCatalog

MakeCatalog will make a catalog from an input image and at least on labeled image. The executable name is `astmkcatalog` with the following general template

```
$ astmkcatalog [OPTION ...] InputImage.fits
```

One line examples:

```
$ astmkcatalog -mdri input.fits
$ astmkcatalog --floatprecision=5 input.fits
$ astmkcatalog --output=cat input_labeled.fits
$ astmkcatalog --objlabs=K_labeled.fits --objhdu=1 \
               --clumplabs=K_labeled.fits --clumphdu=2 i_band.fits
```

If MakeCatalog is to do processing, an input image should be provided with the recognized extensions as input data, see Section 4.1.2 [Arguments], page 46. Optionally a mask file can be specified to ignore some of the pixels in the image, see Section 6.5.3 [Mask image], page 125. Note that if you generated the labeled image using NoiseChisel with a mask image as input, there is no more need to inform MakeCatalog of the mask image, see Section 7.2.1.2 [NoiseChisel output], page 144. The options common to all Gnuastro programs are explained in Section 4.1.4 [Common options], page 48.

MakeCatalog needs 4 (or 5) images as input. These images can be separate extensions in one file (NoiseChisel's default output), or each can have its own file and its own extension. The full 5 images are listed in Section 7.2.1.2 [NoiseChisel output], page 144. However, the clump labels image is not mandatory (when no clump catalog is required, for example in aperture photometry). When inspecting the object labels image, MakeProfiles will look for a `WCLUMPS` (short for with-clumps) header keyword. If that keyword is present and has a value of `yes` (not case sensitive, so `Yes`, or `YES` are also acceptable) then a clump image must also be provided and a clump catalog will be made. When `WCLUMPS` isn't present or has any other value, only an object catalog will be created and all clump related options will be ignored.

For example, if you only need an object catalog from NoiseChisel's output, you can use Header (see Section 5.1 [Header], page 62) to modify or remove the `WCLUMPS` keyword in the objects HDU, then run MakeCatalog on it. Another example can be aperture photometry: let's assume you have made your labeled image (defining the apertures) with MakeProfiles. Clumps are not defined in this context, so besides the input and labeled image, you only need NoiseChisel's Sky and Sky standard deviation images. Since MakeProfile's output

doesn't contain the `WCLUMPS` keyword, you just have to specify your labeled image with the `--objlabs` option and also set its HDU and no clumps catalog will be created. Note that labeled images have to be an integer type. Therefore, when using MakeProfiles to define the apertures/labels, you can use `--type=long` for example.

When a clump catalog is also desired, two catalogs will be made: one for the objects (suffixed with `_o.txt`) and another for the clumps (suffixed with `_c.txt`). Therefore if any value is given to the `--output` option, MakeCatalogs will replace these two suffixes with any existing suffix in the given value. If no output value is given, MakeCatalog will use the input name, see Section 4.5 [Automatic output], page 57. When only building an object catalog, any suffix in the value to output will just be replaced with `.txt`.

The first set of options specify the properties of the inputs. Other necessary input images are treated very much like a mask image, see Section 6.5.3 [Mask image], page 125. If no name is specified for them, their HDU is checked and if that differs from the input HDU, then there is no need to specify a file name for them. The object and column label images or segmentation maps should not be of a floating point type (`BITPIX`).

`-O`
`--objlabs`
(`=STR`) The file name of the object labels, if in the same file as input, it is not mandatory.

`--objhdu` (`=STR`) The HDU of the object labels image, the header keyword `NOBJS` must be present in this extension. The value to this keyword is used as the final number of objects and the number of rows in the output objects catalog. Only pixels with values above zero will be considered.

`-c`
`--clumplabs`
(`=STR`) Similar to `--objlabs` but for the labels of the clumps.

`--clumphdu`
(`=STR`) Similar to `--objhdu`, but for the clumps. The `NCLUMPS` keyword in this header specifies the number of recognized clumps.

`-s`
`--skyfilename`
(`=STR`) File name of an image keeping the Sky value for each pixel.

`--skyhdu` (`=STR`) The HDU of the Sky value image.

`-t`
`--stdfilename`
(`=STR`) File name of image keeping the Sky value standard deviation for each pixel.

`--stdhdu` (`=STR`) The HDU of the Sky value standard deviation image.

`-z`
`--zeropoint`
(`=FLT`) The zero point magnitude for the input image, see Section 8.1.3 [Flux Brightness and magnitude], page 167.

`-E`
`--skysubtracted`
: If the image has already been sky subtracted by another program, then you need to notify MakeCatalog through this option. Note that this is only relevant when the Signal to noise ratio is to be calculated.

`-T`
`--threshold`
: (`=FLT`) For all the columns, only consider pixels that are above a given relative threshold. Symbolizing the value of this option as T, the Sky for a pixel at (i, j) with μ_{ij} and its Standard deviation with σ_{ij}, that pixel will only be used if its value (B_{ij}) satisfies this condition: $B_{ij} > \mu_{ij} + T\sigma_{ij}$. The only calculations that will not be affected are is the average river values (`--riverave`), since they are used as a reference. A commented row will be added in the header of the output catalog that will print the given value, since this is a very important issue, it starts with `**IMPORTANT**`.

  NoiseChisel will detect very diffuse signal which is useful in most cases where the aggregate properties of the detections are desired, since there is signal there (with the desired certainty). However, in some cases, only the properties of the peaks of the objects/clumps are desired, for example in attempting to separate stars from galaxies, the peaks are the major target and the diffuse regions only act to complicate the separation. With this option, MakeCatalog will simply ignore any pixel below the relative threshold.

  This option is not mandatory, so if it isn't given (after reading the command-line and all configuration files, see Section 4.2 [Configuration files], page 51), MakeCatalog will still operate. However, if it has a value in any lower-level configuration file and you want to ignore that value for this particular run or in a higher-level configuration file, then set it to NaN, for example `--threshold=nan`. Gnuastro uses the C library's `strtod` function to read floats, which is not case-sensative in reading NaN values. But to be consistent, it is good practice to only use `nan`.

Through the next group of options, you can customize the general output plain text catalog. The basic idea behind the options about the width and precision of the different types is the fact that some columns don't need too much space, while some do. The width is the number of text columns given to data of each type in the output plain text catalog.

The precision is the number of digits to show after the decimal point in floating point numbers. We have defined two types of floating point numbers here, one is for less accurate precision, like magnitude, while the other is used when more accuracy is necessary. A common example of the latter is right ascension and declination. These variables usually need to be printed with more than 6 point accuracy. Note that all the values are calculated and stored internally as double precision floating point numbers, the distinction made here is only for printing them.

`--nsigmag`
: (`=FLT`) The magnitude of the value to this option multiplied by the maximum standard deviation over the objects or clumps is reported in the output catalog. This value is a per-pixel value, not per object and is not found over an area or

aperture, like the common 5σ values that are commonly reported as a measure of depth. They are based on a certain area and are relics from the time of analog data collection and processing. Modern tools use digital imaging detectors and the area is that of a pixel.

`--intwidth`
(`=INT`) The width of printing the integer values. In MakeCatalog, all IDs, numbers (counters) and areas are considered to be an integer.

`--floatwidth`
(`=INT`) The width of a normal precision floating point column. Any column that is not designated in `--intwidth` or `--accuwidth` is considered to be a normal precision floating point.

`--accuwidth`
(`=INT`) The width columns to be printed with extra accuracy. In MakeCatalog the following columns are printed with extra accuracy: right ascension, declination, brightness, river pixel averages (see Akhlaghi and Ichikawa 2015 for the definition of river pixels), the sky and the sky standard deviation.

`--floatprecision`
(`=INT`) The number of digits to the right of the decimal point in normal floating point display.

`--accuprecision`
(`=INT`) The number of digits to the right of the decimal point in more accurate floating point display.

The upper limit magnitude was discussed in Section 7.3.1 [Quantifying data limits], page 146. Unlike other measured values/columns in MakeCatalog, the upper limit magnitude needs several defined parameters which are discussed here. All the upper limit magnitude specific options start with `up` for upper limit, except for `--envseed` that is also present in other programs and is general for any job requiring random number generation.

One very important consideration in Gnuastro is reproducibility. Therefore, the values to all of these parameters along with others (like the random number generator type and seed) are also reported in the comments of the final catalog when the upper limit magnitude column is desired. One of those parameters is the number of threads used in this run of MakeCatalog. This might seem irrelevant.

The job of sampling a large number of positions over the image is an "embarrasingly parallel" kind of problem which can greatly benefit from threads. For maximum efficiency, the objects in the catalog are divided between the threads and the random number generator is used independently on each thread. So even when the random number generator type and seed are identical, if the number of threads differ, you will get different results. Ofcourse, the difference is due to scatter and is statistically negligible, but it is not exactly reproducible (which is important in some contexts).

`--upmask` (`=STR`) File name of mask image to use for upper-limit calculation. In some cases (especially when doing matched photometry), the object labels specified in the main input and mask image might not be adequate. In other words they do not necessarily have to cover *all* detected objects: the user might have

selected only a few of the objects in their labeled image. All the non-zero pixels of the image specified by this option (in the `--upmaskhdu` extension) will be set as blank for the upper limit magnitude calculation.

For example, when you are using labels from another image, you can give NoiseChisel's objects image output for this image as the value to this option. In this way, you can be sure that regions with data do not harm your distribution. See Section 7.3.1 [Quantifying data limits], page 146, for more on the upper limit magnitude.

`--upmaskhdu`
(`=STR`) The extension in the file specified by `--upmask`.

`--upnum` (`=INT`) The number of random samples to take for all the objects. A larger value to this option will give a more accurate result (assymptotically), but it will also slow down the process. When a randomly positioned sample overlaps with a detected/masked pixel it is not counted and another random position is found until the object completely lies over an undetected region. So you can be sure that for each object, this many samples over undetected objects are made. See the upper limit magnitude discussion in Section 7.3.1 [Quantifying data limits], page 146, for more.

`--envseed`
Read the random number generator type and seed value from the environment (see Section 8.2.1.4 [Generating random numbers], page 178). Random numbers are used in calculating the random positions of different samples of each object.

`--upsclipmultip`
(`=FLT`) The multiple of σ to use in σ-clipping the final distribution (see Section 7.1.2 [Sigma clipping], page 129). The images might have some artifacts or some regions with signal in the image might not have been removed correctly. If a random sampling falls over such regions, they can significantly bias the final standard deviation. To avoid the effect of such outliers, after the distribution is found, it is σ-clipped (by convergence).

`--upsclipaccu`
(`=FLT`) Relative difference between two successive σ measurements to halt the σ-clipping process by convergence, see the explanations for `--upsclipmultip` for more.

`--upnsigma`
(`=FLT`) The multiple of the standard deviation (or σ) used to measure the magnitude. Note that this is the final σ produced after the σ-clipping.

The final group of options particular to MakeCatalog are those that specify which columns should be displayed in the output catalogs. For each column there is an option, if it has been called on the command line or in any of the configuration files, it will included as a column in the output catalog. Some of the options apply to both objects and clumps and some are particular to only one of them. The latter cases are explicitly marked with [Objects] or [Clumps] to specify the catalog they will be placed in.

The order of the columns in the output catalog is the inverse of the order their options are read in. For example see the following command[6]:

```
$ astmkcatalog --magnitude --dec --ra --id input.fits
```

In this example, if no columns are specified in any of the configuration files (see Section 4.2 [Configuration files], page 51), then the columns in the catalogs produced by this command will have the following order: ID, RA, Dec and Magnitude. Contrary to what it looks like, this is done to make life easier for the users. The configuration files can also keep any of the columns (so you don't have to specify your desired columns every time). This inverse ordering thus comes from their precedence, see Section 4.2.2 [Configuration file precedence], page 52.

For example catalogs usually have at least an ID column and position columns (in the image and/or the world coordinate system). By reading the order of the columns in reverse you can have your fixed set of columns in your system wide configuration file and in any particular run, if you want some other information about objects or clumps, you can add those columns on the command-line. Through the user and current directory configuration files, you can also have custom catalogs in each of your working directories, without bothering to specify the columns every time you run MakeCatalog in those directories.

`--i`
`--id` The ID of the clump or object.

`-j`
`hostobjid`
[Clumps] The ID of the object which hosts this clump.

`-I`
`--idinhostobj`
[Clumps] The ID of this clump in its host object.

`-C`
`--numclumps`
[Objects] The number of clumps in this object.

`-a`
`--area` The area (number of pixels) in any clump or object.

`--clumpsarea`
[Objects] The total area of all the clumps in this object.

`-x`
`--x` The flux weighted center of all objects and clumps along the first FITS axis (horizontal when viewed in SAO ds9). The weight has to have a positive value (pixel value larger than the Sky value) to be meaningful! Specially when doing matched photometry, this might not happen: no pixel value might be above the Sky value. In such cases, where there was no pixel with a value larger than the Sky in the object or clump, the geometric center (ignoring pixel values, only their positions) over the object or clump is used instead of the flux weighted center (see `--geox`).

[6] This command is practically identical to the first command in the one-line examples, see Section 4.1.3 [Options], page 46, for an explanation of the concatenation of the on/off options.

`-y`
`--y` The flux weighted center of all objects and clumps along the second FITS axis (vertical when viewed in SAO ds9). See `--x`.

`--geox` The geometric center of all objects and clumps along the first FITS axis axis. The geometric center is the average pixel positions irrespective of their pixel values.

`--geoy` The geometric center of all objects and clumps along the second FITS axis axis, see `--geox`.

`--clumpsx`
[Objects] The flux weighted center of all the clumps in this object along the first FITS axis. See `--x`.

`--clumpsy`
[Objects] The flux weighted center of all the clumps in this object along the second FITS axis. See `--x`.

`--clumpsgeox`
[Objects] The geometric center of all the clumps in this object along the first FITS axis. See `--geox`.

`--clumpsgeoy`
[Objects] The geometric center of all the clumps in this object along the second FITS axis. See `--geox`.

`-r`
`--ra` Flux weighted right ascension of all objects or clumps, see `--x`.

`-d`
`--dec` Flux weighted declination of all objects or clumps, see `--x`.

`--georа` Geometric center right ascension of all objects or clumps, see `--geox`.

`--geodec` Geometric center declination of all objects or clumps, see `--geox`.

`--clumpsra`
[Objects] Flux weighted right ascension of all clumps in this object, see `--x`.

`--clumpsdec`
[Objects] Flux weighted declination of all clumps in this object, see `--x`.

`--clumpsgeora`
[Objects] Geometric center right ascension of all clumps in this object, see `--geox`.

`--clumpsgeodec`
[Objects] Geometric center declination of all clumps in this object, see `--geox`.

`-b`
`--brightness`
The brightness (sum of all pixel values), see Section 8.1.3 [Flux Brightness and magnitude], page 167. For each clump in the clumps catalog, the ambient brightness (flux of river pixels around the clump multiplied by the area of

the clump) is removed, see `--riverflux`. So the sum of clump brightnesses in the clump catalog will be smaller than the total clump brightness in the `--clumpbrightness` column of the objects catalog.

`--clumpbrightness`
: [Objects] The total brightness of the clumps within an object. This is simply the sum of the pixels associated with clumps in the object.

`--noriverbrightness`
: [Clumps] The Sky (not river) subtracted clump brightness. By definition, for the clumps, the average brightness of the rivers surrounding it are subtracted from it for a first order accounting for contamination by neighbors. In cases where you will be calculating the flux brightness difference later (one example below) the contamination will be (mostly) removed at that stage, which is why this column was added.

  One example might be this: you want to know the change in the clump flux as a function of threshold (see `--threshold`). So you will make two catalogs (each having this column but with different thresholds) and then subtract the lower threshold catalog (higher brightness) from the higher threshold catalog (lower brightness). The effect is most visible when the rivers have a high average signal-to-noise ratio. The removed contribution from the pixels below the threshold will be less than the river pixels. Therefore the river-subtracted brightness (`--brightness`) for the thresholded catalog for such clumps will be larger than the brightness with no threshold!

`-m`
`--magnitude`
: The magnitude of clumps or objects, see `--brightness`.

`-e`
`--magnitudeerr`
: The magnitude error of clumps or objects. The magnitude error is calculated from the signal-to-noise ratio (see `--sn`, abbreviated as S): $\Delta M = 2.5/(S \times \ln 10)$. Note that until now this error assumes un-correlated pixel values and also does not include the error in estimating the aperture (or error in generating the labeled image).

  For now these factors have to be found by other means. Task 14124 (`https://savannah.gnu.org/task/index.php?14124`) has been defined for work on adding these sources of error too.

`--clumpsmagnitude`
: [Objects] The magnitude of all clumps in this object, see `--clumpbrightness`.

`--upperlimitmag`
: [Objects] The upper limit magnitude for this object. The object's footprint is used over randomly positioned parts of the image that do not cover a detected object. The standard deviation of the final distribution is then the upper limit magnitude, see Section 7.3.1 [Quantifying data limits], page 146, for a complete explanation. This is very important for the fainter objects in the image where the measured magnitudes are not reliable.

`--riverave`
: [Clumps] The average brightness of the river pixels around this clump. River pixels were defined in Akhlaghi and Ichikawa 2015. In short they are the pixels immediately outside of the clumps. This value is used internally to find the brightness (or magnitude) and signal to noise ratio of the clumps. It can generally also be used as a scale to gauge the base (ambient) flux surrounding the clump. In case there was no river pixels, then this column will have the value of the Sky under the clump. So note that this value is *not* sky subtracted.

`--rivernum`
: [Clumps] The number of river pixels around this clump, see `--riverflux`.

`-n`
`--sn`
: The Signal to noise ratio (S/N) of all clumps or objects. See Akhlaghi and Ichikawa (2015) for the exact equations used.

`--sky`
: The sky flux (per pixel) value under this object or clump. This is actually the mean value of all the pixels in the sky image that lie on the same position as the object or clump.

`--std`
: The sky value standard deviation (per pixel) for this clump or object. Like `--sky`, this is the average of the values in the input sky standard deviation image pixels that lie over this object.

`-A`
`--semimajor`
: The pixel-value weighted semi-major axis of the profile (assuming it is an ellipse) in units of pixels.

`-B`
`--semiminor`
: The pixel-value weighted semi-minor axis of the profile (assuming it is an ellipse) in units of pixels.

`-p`
`--positionangle`
: The pixel-value weighted angle of the semi-major axis with the first FITS axis in degrees.

`--geosemimajor`
: The geometric (ignoring pixel values) semi-major axis of the profile, assuming it is an ellipse.

`--geosemiminor`
: The geometric (ignoring pixel values) semi-minor axis of the profile, assuming it is an ellipse.

`--geopositionangle`
: The geometric (ignoring pixel values) angle of the semi-major axis with the first FITS axis in degrees.

7.3.4 Adding new columns to MakeCatalog

The common development characteristics of MakeCatalog and other Gnuastro programs is explained in Chapter 11 [Developing], page 236. This section might be more clearly understood after reading that chapter. MakeCatalog has been designed to allow easy addition of new columns, here we will give a fast over-view of the steps you need to take to define a new output column. After adding and testing your column, you are most welcome (and encouraged) to share it with us so we can add to the next release of Gnuastro for everyone else to also benefit from your efforts.

MakeCatalog will first have two passes over the input pixels: in the first pass it will gather mainly object information and in the second run, it will mainly focus on the clumps, or any other measurement that needs an output from the first pass. These two passes are designed to be raw summations: no extra processing. This will allow parallel processing and simplicity/clarity. Once the passes are done, depending on the order of options that the user has defined the appropriate function will be called to add a processed column from the raw calculations to the output array (which will be printed). So if your new calculation, needs new raw information from the pixels, then you will need to also modify the respective `firstpass` and `secondpass` functions and define new information table columns in `main.h`.

For all the steps below, it is easiest to just copy and paste an existing option and change the variables in each step. In all these different places, the options are sorted in the same order (same order as Section 7.3.3 [Invoking MakeCatalog], page 151). This allows a particular column/option to be easily found in all steps. Therefore in adding your new option, be sure to keep it in the same relative place in the list in all the separate places (it doesn't necessarily have to be in the end), and near conceptually similar options.

`main.h` The `objectcols` and `clumpcols` enumerated variables (`enum`) define the labels for the separate information table columns (the raw pixel calculations). If your new calculation requires new raw columns, the add new macros for them. The `outcols` enumerated variables define the recognized output columns. You will need to add a new name for your desired column here, then add a row for your new parameter in the `uiparams` structure, the name should be the same as the one you used in `outcols` (in small caps).

`args.h` This file specifies all the parameters for the GNU C library, Argp structure that is in charge of reading the user's options. It is best that the option name be the same as the name you put in `outcols` (in `main.h`), like the other options that are already defined.

Define a new option in `argp_option` (rows between the `{` and `}`). The first is the name of the option, the second is its short option name (see Section 4.1.3 [Options], page 46). The next two values should be `0` and the last should be a short description of your option/column. For the second element is best to use a number (so no short option is defined for the user) which in Gnuastro is defined to be a number larger than 500. The range of used numbers (and short option names) are shown in the comment immediately above `argp_option` structure, increase that number by one and use that number for your new option.

Go down to the `parse_opt` function and add a `case` for your specific option (using the same number you specified above), then update the contents, based on the new macros you defined `main.h`.

`ui.c` In the `readconfig` function, add an `else if` for your option (easiest to just copy and paste, then update). In the `printvalues` function's `switch` structure, add a `case` for your option. Finally, in the `preparearrays` function, add a `case` for your new output column. Note that if the column is to be used for both objects and clumps, you need to set both `p->objcols` and `p->clumpcols`.

`mkcatalog.c`
If your procedure needs new raw calculations, add a new line to appropriate parts of the `firstpass`, and/or `secondpass` functions, they should be clear and are fully commented. Then add a `case` for your column in the `makeoutput` function, here you will need to call a function (which should be defined in `columns.c`).

`columns.c`
Define the function that will write your desired column in the output column using the raw input calculations here. There are a lot of functions in this file which you can use to model and they are commented, in short you have to specify a header (commented column), the units of your variable, the possible accuracy to print and finally the actual columns.

8 Modeling and fitting

In order to fully understand observations after initial analysis on the image, it is very important to compare them with the existing models to be able to further understand both the models and the data. The tools in this chapter create model galaxies and will provide 2D fittings to be able to understand the detections.

8.1 MakeProfiles

MakeProfiles will create mock astronomical profiles from a catalog, either individually or together in one output image. In data analysis, making a mock image can act like a calibration tool, through which you can test how successfully your detection technique is able to detect a known set of objects. There are commonly two aspects to detecting: the detection of the fainter parts of bright objects (which in the case of galaxies fade into the noise very slowly) or the complete detection of an over-all faint object. Making mock galaxies is the most accurate (and idealistic) way these two aspects of a detection algorithm can be tested. You also need mock profiles in fitting known functional profiles with observations.

MakeProfiles was initially built for extra galactic studies, so currently the only astronomical objects it can produce are stars and galaxies. We welcome the simulation of any other astronomical object. The general outline of the steps that MakeProfiles takes are the following:

1. Build the full profile out to its truncation radius in a possibly over-sampled array.
2. Multiply all the elements by a fixed constant so its total magnitude equals the desired total magnitude.
3. If `--individual` is called, save the array for each profile to a FITS file.
4. If `--nomerged` is not called, add the overlapping pixels of all the created profiles to the output image and abort.

Using input values, MakeProfiles adds the World Coordinate System (WCS) headers of the FITS standard to all its outputs (except PSF images!). For a simple test on a set of mock galaxies in one image, there is no need for the third step or the WCS information.

However in complicated simulations like weak lensing simulations, where each galaxy undergoes various types of individual transformations based on their position, those transformations can be applied to the different individual images with other programs. After all the transformations are applied, using the WCS information in each individual profile image, they can be merged into one output image for convolution and adding noise.

8.1.1 Modeling basics

In the subsections below, first a review of some very basic information and concepts behind modeling a real astronomical image is given. You can skip this subsection if you are already sufficiently familiar with these concepts.

8.1.1.1 Defining an ellipse

The PSF, see Section 8.1.1.2 [Point Spread Function], page 163, and galaxy radial profiles are generally defined on an ellipse so in this section first defining an ellipse on a pixelated 2D surface is discussed. Labeling the major axis of an ellipse a, and its minor axis with b,

the axis ratio is defined as: $q \equiv b/a$. The major axis of an ellipse can be aligned in any direction, therefore define the angle of the major axis to the horizontal axis of the image is defined to be the position angle of the ellipse and in this book, we show it with θ.

Our aim is to put a radial profile of any functional form $f(r)$ over an ellipse. Let's define the radial distance r_{el} as the distance on the major axis to the center of the ellipse which is located at x_c and y_c. We want to find the elliptical distance of a point located at (i, j), in the image coordinate system, from the center of the ellipse. First the coordinate system is rotated by θ to get the new rotated coordinates of that point (i_r, j_r):

$$i_r(i,j) = (i_c - i)\cos(\theta) + (j_c - j)\sin(\theta)$$

$$j_r(i,j) = (j_c - j)\cos(\theta) - (i_c - i)\sin(\theta)$$

The elliptical distance of a point located at (i, j) can now be defined as: $r_{el}^2 = \sqrt{i_r^2 + j_r^2/q^2}$. To place the radial profiles explained below over an ellipse, $f(r_{el}(i, j))$ is calculated based on the functional radial profile desired.

The way MakeProfiles builds the profile is that the nearest pixel in the image to the given profile center is found and the profile value is calculated for it, see Section 8.1.1.5 [Sampling from a function], page 166. The next pixel which the profile value is calculated on is the next nearest neighbor of the initial pixel to the profile center (as defined by r_{el}). This is done fairly efficiently using a breadth first parsing strategy[1] which is implemented through an ordered linked list.

Using this approach, we only go over one layer of pixels on the circumference of the profile to build the profile. Not one more extra pixel has to be checked. Another consequence of this strategy is that extending MakeProfiles to three dimensions becomes very simple: only the neighbors of each pixel have to be changed. Everything else after that (when the pixel index and its radial profile have entered the linked list) is the same, no matter the number of dimensions we are dealing with.

8.1.1.2 Point Spread Function

Assume we have a 'point' source, or a source that is far smaller than the maximum resolution (a pixel). When we take an image of it, it will 'spread' over an area. To quantify that spread, we can define a 'function'. This is how the point spread function or the PSF of an image is defined. This 'spread' can have various causes, for example in ground based astronomy, due to the atmosphere. In practice we can never surpass the 'spread' due to the diffraction of the lens aperture. Various other effects can also be quantified through a PSF. For example, the simple fact that we are sampling in a discrete space, namely the pixels, also produces a very small 'spread' in the image.

Convolution is the mathematical process by which we can apply a 'spread' to an image, or in other words blur the image, see Section 6.3.1.1 [Convolution process], page 89. The Brightness of an object should remain unchanged after convolution, see Section 8.1.3 [Flux Brightness and magnitude], page 167. Therefore, it is important that the sum of all the pixels of the PSF be unity. The PSF image also has to have an odd number of pixels on its

[1] `http://en.wikipedia.org/wiki/Breadth-first_search`

sides so one pixel can be defined as the center. In MakeProfiles, the PSF can be set by the two methods explained below.

Parametric functions

A known mathematical function is used to make the PSF. In this case, only the parameters to define the functions are necessary and MakeProfiles will make a PSF based on the given parameters for each function. In both cases, the center of the profile has to be exactly in the middle of the central pixel of the PSF (which is automatically done by MakeProfiles). When talking about the PSF, usually, the full width at half maximum or FWHM is used as a scale of the width of the PSF.

Gaussian In the older papers, and to a lesser extent even today, some researchers use the 2D Gaussian function to approximate the PSF of ground based images. In its most general form, a Gaussian function can be written as:

$$f(r) = a \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) + d$$

Since the center of the profile is pre-defined, μ and d are constrained. a can also be found because the function has to be normalized. So the only important parameter for MakeProfiles is the σ. In the Gaussian function we have this relation between the FWHM and σ:

$$\mathrm{FWHM_g} = 2\sqrt{2\ln 2}\sigma \approx 2.35482\sigma$$

Moffat The Gaussian profile is much sharper than the images taken from stars on photographic plates or CCDs. Therefore in 1969, Moffat proposed this functional form for the image of stars:

$$f(r) = a\left[1 + \left(\frac{r}{\alpha}\right)^2\right]^{-\beta}$$

Again, a is constrained by the normalization, therefore two parameters define the shape of the Moffat function: α and β. The radial parameter is α which is related to the FWHM by

$$\mathrm{FWHM_m} = 2\alpha\sqrt{2^{1/\beta} - 1}$$

Comparing with the PSF predicted from atmospheric turbulence theory with a Moffat function, Trujillo et al.[2] claim that β should

[2] Trujillo, I., J. A. L. Aguerri, J. Cepa, and C. M. Gutierrez (2001). "The effects of seeing on Sérsic profiles - II. The Moffat PSF". In: MNRAS 328, pp. 977—985.

be 4.765. They also show how the Moffat PSF contains the Gaussian PSF as a limiting case when $\beta \to \infty$.

An input FITS image
: An input image file can also be specified to be used as a PSF. If the sum of its pixels are not equal to 1, the pixels will be multiplied by a fraction so the sum does become 1.

While the Gaussian is only dependent on the FWHM, the Moffat function is also dependent on β. Comparing these two functions with a fixed FWHM gives the following results:

- Within the FWHM, the functions don't have significant differences.
- For a fixed FWHM, as β increases, the Moffat function becomes sharper.
- The Gaussian function is much sharper than the Moffat functions, even when β is large.

8.1.1.3 Stars

In MakeProfiles, stars are generally considered to be a point source. This is usually the case for extra galactic studies, were nearby stars are also in the field. Since a star is only a point source, we assume that it only fills one pixel prior to convolution. In fact, exactly for this reason, in astronomical images the light profiles of stars are one of the best methods to understand the shape of the PSF and a very large fraction of scientific research is preformed by assuming the shapes of stars to be the PSF of the image.

8.1.1.4 Galaxies

Today, most practitioners agree that the flux of galaxies can be modeled with one or a few generalized de Vaucouleur's (or Sérsic) profiles.

$$I(r) = I_e \exp\left(-b_n\left[\left(\frac{r}{r_e}\right)^{1/n} - 1\right]\right)$$

Gérard de Vaucouleurs (1918-1995) was first to show in 1948 that this function best fits the galaxy light profiles, with the only difference that he held n fixed to a value of 4. 20 years later in 1968, J. L. Sérsic showed that n can have a variety of values and does not necessarily need to be 4. This profile depends on the effective radius (r_e) which is defined as the radius which contains half of the profile brightness (see Section 8.1.4 [Profile magnitude], page 168). I_e is the flux at the effective radius. The Sérsic index n is used to define the concentration of the profile within r_e and b_n is a constant dependent on n. MacArthur et al.[3] show that for $n > 0.35$, b_n can be accurately approximated using this equation:

$$b_n = 2n - \frac{1}{3} + \frac{4}{405n} + \frac{46}{25515n^2} + \frac{131}{1148175n^3} - \frac{2194697}{30690717750n^4}$$

[3] MacArthur, L. A., S. Courteau, and J. A. Holtzman (2003). "Structure of Disk-dominated Galaxies. I. Bulge/Disk Parameters, Simulations, and Secular Evolution". In: ApJ 582, pp. 689—722.

8.1.1.5 Sampling from a function

A pixel is the ultimate level of accuracy to gather data, we can't get any more accurate in one image, this is known as sampling in signal processing. However, the mathematical profiles which describe our models have infinite accuracy. Over a large fraction of the area of astrophysically interesting profiles (for example galaxies or PSFs), the variation of the profile over the area of one pixel is not too significant. In such cases, the elliptical radius (r_{el} of the center of the pixel can be assigned as the final value of the pixel, see Section 8.1.1.1 [Defining an ellipse], page 162).

As you approach their center, some galaxies become very sharp (their value significantly changes over one pixel's area). This sharpness increases with smaller effective radius and larger Sérsic values. Thus rendering the central value extremely inaccurate. The first method that comes to mind for solving this problem is integration. The functional form of the profile can be integrated over the pixel area in a 2D integration process. However, unfortunately numerical integration techniques also have their limitations and when such sharp profiles are needed they can become extremely inaccurate.

The most accurate method of sampling a continuous profile on a discrete space is by choosing a large number of random points within the boundaries of the pixel and taking their average value (or Monte Carlo integration). This is also, generally speaking, what happens in practice with the photons on the pixel. The number of random points can be set with `--numrandom`.

Unfortunately, repeating this Monte Carlo process would be extremely time and CPU consuming if it is to be applied to every pixel. In order to not loose too much accuracy, in MakeProfiles, the profile is built using both methods explained above. The building of the profile begins from its central pixel and continues outwards. Monte Carlo integration is first applied (which yields F_r), then the central pixel value (F_c) on the same pixel. If the fractional difference ($|F_r - F_c|/F_r$) is lower than a given tolerance level we will stop using Monte Carlo integration and only use the central pixel value.

The ordering of the pixels in this inside-out construction is based on $r = \sqrt{(i_c - i)^2 + (j_c - j)^2}$, not r_{el}, see Section 8.1.1.1 [Defining an ellipse], page 162. When the axis ratios are large (near one) this is fine. But when they are small and the object is highly elliptical, it might seem more reasonable to follow r_{el} not r. The problem is that the gradient is stronger in pixels with smaller r (and larger r_{el}) than those with smaller r_{el}. In other words, the gradient is strongest along the minor axis. So if the next pixel is chosen based on r_{el}, the tolerance level will be reached sooner and lots of pixels with large fractional differences will be missed.

Monte Carlo integration uses a random number of points. Thus, every time you run it, by default, you will get a different distribution of points to sample within the pixel. In the case of large profiles, this will result in a slight difference of the pixels which use Monte Carlo integration each time MakeProfiles is run. To have a deterministic result, you have to fix the random number generator properties which is used to build the random distribution. This can be done by setting the `GSL_RNG_TYPE` and `GSL_RNG_SEED` environment variables and calling MakeProfiles with the `--envseed` option. To learn more about the process of generating random numbers, see Section 8.2.1.4 [Generating random numbers], page 178.

The seed values are fixed for every profile: with `--envseed`, all the profiles have the same seed and without it, each will get a different seed using the system clock (which is

accurate to within one microsecond). The same seed will be used to generate a random number for all the sub-pixel positions of all the profiles. So in the former, the sub-pixel points checked for all the pixels undergoing Monte carlo integration in all profiles will be identical. In other words, the sub-pixel points in the first (closest to the center) pixel of all the profiles will be identical with each other. All the second pixels studied for all the profiles will also receive an identical (different from the first pixel) set of sub-pixel points and so on. As long as the number of random points used is large enough or the profiles are not identical, this should not cause any systematic bias.

8.1.1.6 Oversampling

The steps explained in Section 8.1.1.5 [Sampling from a function], page 166, do give an accurate representation of a profile prior to convolution. However, in an actual observation, the image is first convolved with or blurred by the atmospheric and instrument PSF in a continuous space and then it is sampled on the discrete pixels of the camera.

In order to more accurately simulate this process, the unconvolved image and the PSF are created on a finer pixel grid. In other words, the output image is a certain odd-integer multiple of the desired size, we can call this 'oversampling'. The user can specify this multiple as a command-line option. The reason this has to be an odd number is that the PSF has to be centered on the center of its image. An image with an even number of pixels on each side does not have a central pixel.

The image can then be convolved with the PSF (which should also be oversampled on the same scale). Finally, image can be sub-sampled to get to the initial desired pixel size of the output image. After this, mock noise can be added as explained in the next section. This is because unlike the PSF, the noise occurs in each output pixel, not on a continuous space like all the prior steps.

8.1.2 If convolving afterwards

In case you want to convolve the image later with a given point spread function, make sure to use a larger image size. After convolution, the profiles become larger and a profile that is normally completely outside of the image might fall within it.

On one axis, if you want your final (convolved) image to be m pixels and your PSF is $2n+1$ pixels wide, then when calling MakeProfiles, set the axis size to $m+2n$, not m. You also have to shift all the pixel positions of the profile centers on the that axis by n pixels to the positive.

After convolution, you can crop the outer n pixels with the section crop box specification of ImageCrop: `--section=n:*-n,n:*-n` assuming your PSF is a square, see Section 6.1.2 [Crop section syntax], page 77. This will also remove all discrete Fourier transform artifacts (blurred sides) from the final image. To facilitate this shift, MakeProfiles has the options `--xshift`, `--yshift` and `--prepforconv`, see Section 8.1.5 [Invoking MakeProfiles], page 169.

8.1.3 Flux Brightness and magnitude

Astronomical data pixels are usually in units of counts[4] or electrons or either one divided by seconds. To convert from the counts to electrons, you will need to know the instrument

[4] Counts are also known as analog to digital units (ADU).

gain. In any case, they can be directly converted to energy or energy/time using the basic hardware (telescope, camera and filter) information. We will continue the discussion assuming the pixels are in units of energy/time.

The *brightness* of an object is defined as its total detected energy per time. This is simply the sum of the pixels that are associated with that detection by our detection tool for example Section 7.2 [NoiseChisel], page 136[5]. The *flux* of an object is in units of energy/time/area and for a detected object, it is defined as its brightness divided by the area used to collect the light from the source or the telescope aperture (for example in cm^2)[6]. Knowing the flux (f) and distance to the object (r), we can calculate its *luminosity*: $L = 4\pi r^2 f$. Therefore, flux and luminosity are intrinsic properties of the object, while brightness depends on our detecting tools (hardware and software). Here we will not be discussing luminosity, but brightness. However, since luminosity is the astrophysically interesting quantity, we also defined it here to avoid possible confusion between these two terms because they both have the same units.

Images of astronomical objects span over a very large range of brightness. With the Sun (as the brightest object) being roughly $2.5^{60} = 10^{24}$ times brighter than the faintest galaxies we can currently detect. Therefore discussing brightness will be very hard, and astronomers have chosen to use a logarithmic scale to talk about the brightness of astronomical objects. But the logarithm can only be usable with a unit-less and always positive value. Fortunately brightness is always positive and to remove the units we divide the brightness of the object (B) by a reference brightness (B_r). We then define the resulting logarithmic scale as *magnitude* through the following relation[7]

$$m - m_r = -2.5 \log_{10} \left(\frac{B}{B_r} \right)$$

m is defined as the magnitude of the object and m_r is the pre-defined magnitude of the reference brightness. One particularly easy condition is when $B_r = 1$. This will allow us to summarize all the hardware specific parameters discussed above into one number as the reference magnitude which is commonly known as the Zero-point[8] magnitude.

8.1.4 Profile magnitude

To find the profile brightness or its magnitude, (see Section 8.1.3 [Flux Brightness and magnitude], page 167), it is customary to use the 2D integration of the flux to infinity. However, in MakeProfiles we do not follow this idealistic approach and apply a more realistic method to find the total brightness or magnitude: the sum of all the pixels belonging to a profile within its predefined truncation radius. Note that if the truncation radius is not large enough, this can be significantly different from the total integrated light to infinity.

[5] If further processing is done, for example the Kron or Petrosian radii are calculated, then the detected area is not sufficient and the total area that was within the respective radius must be used.

[6] For a full object that spans over several pixels, the telescope area should be used to find the flux. However, sometimes, only the brightness per pixel is desired. In such cases this book also *loosely* uses the term flux. This is only approximately accurate however, since while all the pixels have a fixed area, the pixel size can vary with camera on the telescope.

[7] The -2.5 factor in the definition of magnitudes is a legacy of the our ancient colleagues and in particular Hipparchus of Nicaea (190-120 BC).

[8] When $B = Br = 1$, the right side of the magnitude definition will be zero. Hence the name, "zero-point".

An integration to infinity is not a realistic condition because no galaxy extends indefinitely (important for high Sérsic index profiles), pixelation can also cause a significant difference between the actual total pixel sum value of the profile and that of integration to infinity, especially in small and high Sérsic index profiles. To be safe, you can specify a large enough truncation radius for such compact high Sérsic index profiles.

If oversampling is used then the brightness is calculated using the over-sampled image, see Section 8.1.1.6 [Oversampling], page 167, which is much more accurate. The profile is first built in an array completely bounding it with a normalization constant of unity (see Section 8.1.1.4 [Galaxies], page 165). Taking B to be the desired brightness and S to be the sum of the pixels in the created profile, every pixel is then multiplied by B/S so the sum is exactly B.

If the `--individual` option is called, this same array is written to a FITS file. If not, only the overlapping pixels of this array and the output image are kept and added to the output array.

8.1.5 Invoking MakeProfiles

MakeProfiles will make any number of profiles specified in a catalog either individually or in one image. The executable name is `astmkprof` with the following general template

```
$ astmkprof [OPTION ...] [BackgroundImage] Catalog
```

One line examples:

```
$ astmkprof background.fits catalog.txt
$ astmkprof --xcol=0 --ycol=1 catalog.txt
$ astmkprof --individual --oversample 3 -x500 -y500 catalog.txt
```

If mock images are to be made, the catalog (which stores the parameters for each mock profile) is the mandatory argument. The input catalog has to be a text file formatted in a table with columns separated by space, tab or comma (`,`) characters. See Chapter 4 [Common program behavior], page 45, for a complete explanation of some common behaviour and options in all Gnuastro programs including MakeProfiles.

If a data image file (see Section 4.1.2 [Arguments], page 46) is given, the pixels of that image are used as the background value for every pixel. The flux value of each profile pixel will be added to the pixel in that background value. In this case the values to all options relating to the output size and WCS will be ignored if specified (for example `--naxis1`, `--naxis2` and `--prepforconv`) on the command-line or in the configuration files. Note that `--oversample` will remain active even if a background image is specified.

8.1.5.1 MakeProfiles catalog

The catalog is a text file table. Its columns can be ordered in any desired manner, you can specify which columns belong to which parameters using the set of options ending with `col`, for example `--xcol` or `--rcol`, see Section 8.1.5.2 [MakeProfiles options], page 170.

The value for the profile center in the catalog (in the `--xcol` and `--ycol` columns) can be a floating point number so the profile center can be on any sub-pixel position. Note that pixel positions in the FITS standard start from 1 and an integer is the pixel center. So a 2D image actually starts from the position (0.5, 0.5). In MakeProfiles profile centers do not have to be in the image. Even if only one pixel of the profile within the truncation radius is within the output image, that pixel is included in the image. Profiles that are completely

out of the image will not be created. You can use the output log file to see which profiles were within the image.

If PSF profiles (Moffat or Gaussian) are in the catalog and the profiles are to be built in one image (when `--individual` is not used), it is assumed they are the PSF(s) you want to convolve your created image with. So by default, they will not be built in the output image but as separate files. The sum of pixels of these separate files will also be set to unity (1) so you are ready to convolve, see Section 6.3.1.1 [Convolution process], page 89. As a summary, their position and magnitude will be ignored. This behaviour can be disabled with the `--psfinimg` option. If you want to create all the profiles separately (with `--individual`) and you want the sum of the PSF profile pixels to be unity, you have to set their magnitudes in the catalog to the zero-point magnitude and be sure that the central positions of the profiles don't have any fractional part (the PSF center has to be in the center of the pixel).

8.1.5.2 MakeProfiles options

The common options that are shared by Gnuastro programs, are fully explained in Section 4.1.4 [Common options], page 48, and are not repeated here. Since there are no image inputs, the`--hdu` option is ignored. The options can be classified into the following categories: Output, Profiles, Catalog and WCS. Below each one is reviewed.

Output:

`-x`
`--naxis1` (`=INT`) The number of pixels in the output image along the first FITS axis (horizontal when viewed in SAO ds9). This is before over-sampling. For example if you call MakeProfiles with `--naxis1=100 --oversample=5` (assuming no shift due for later convolution), then the final image size along the first axis will be 500. If a background image is specified, any possible value to this option is ignored.

`-y`
`--naxis2` (`=INT`) The number of pixels in the output image along the second FITS axis (vertical when viewed in SAO ds9), see the explanation for `--naxis1`.

`-T`
`--type` (`=STR`) Type of the output image pixels (specified by `BITPIX` in the FITS standard). It can have any one of the following values: `byte`, `short`, `long`, `longlong`, `float`, `double`. Internally, the images are made in the float type, but upon writing of the single output (not individual profiles), the image will be converted to the type specified by this option.

When a background image is given without the `--inputascanvas` option, it is assumed that you want the output in the same format as the background, therefore the value to this option is ignored.

`-C`
`--inputascanvas`
When an image is given as an argument, use the input image as the canvas to make all the mock profiles. In practice, this option only changes the non-blank pixels (see Section 6.1.3 [Blank pixels], page 77) of the input image (in memory

after reading, not the actual input file) to zero before making the profiles. The profiles are then built on those cleared pixels. With this option (like when building on a background image) the values to `--naxis1`, `--naxis2`, `--crpix1`, `--crpix2`, `--crval1`, `--crval2`, and `--resolution` are ignored. However, if `--type` is given, its value will take precedence over the background image type.

This option can be very useful when the mock image is to be made over a real sky region and compared to a real image (that might have blank pixels. One example can be when you want to make circular or elliptical labeled regions to feed into MakeCatalog along with NoiseChisel's outputs for aperture photometry. See the discussion and examples under the '`--mforflatpix`' for more.

`-s`
`--oversample`
(`=INT`) The scale to over-sample the profiles and final image. If not an odd number, will be added by one, see Section 8.1.1.6 [Oversampling], page 167. Note that this `--oversample` will remain active even if an input image is specified. If your input catalog is based on the background image, be sure to set `--oversample=1`.

`--psfinimg`
Build the possibly existing PSF profiles (Moffat or Gaussian) in the catalog into the final image. By default they are built separately so you can convolve your images with them, thus their magnitude and positions are ignored. With this option, they will be built in the final image like every other galaxy profile. To have a final PSF in your image, make a point profile where you want the PSF and after convolution it will be the PSF.

`-i`
`--individual`
If this option is called, each profile is created in a separate FITS file within the same directory as the output and the row number of the profile (starting from zero) in the name. The file for each row's profile will be in the same directory as the final combined image of all the profiles and will have the final image's name as a suffix. So for example if the final combined image is named `./out/fromcatalog.fits`, then the first profile that will be created with this option will be named `./out/0_fromcatalog.fits`.

Since each image only has one full profile out to the truncation radius the profile is centered and so, only the sub-pixel position of the profile center is important for the outputs of this option. The output will have an odd number of pixels. If there is no oversampling, the central pixel will contain the profile center. If the value to `--oversample` is larger than unity, then the profile center is on any of the central `--oversample`'d pixels depending on the fractional value of the profile center.

If the fractional value is larger than half, it is on the bottom half of the central region. This is due to the FITS definition of a real number position: The center of a pixel has fractional value 0.00 so each pixel contains these fractions: .5 – .75 – .00 (pixel center) – .25 – .5.

`-m`
`--nomerged`
Don't make a merged image. By default after making the profiles, they are added to a final image with sides of `--naxis1` and `--naxis2` if they overlap with it.

Profiles:

`-r`
`--numrandom`
The number of random points used in the central regions of the profile, see Section 8.1.1.5 [Sampling from a function], page 166.

`-e`
`--envseed`
Use the value to the `GSL_RNG_SEED` environment variable to generate the random Monte Carlo sampling distribution, see Section 8.1.1.5 [Sampling from a function], page 166, and Section 8.2.1.4 [Generating random numbers], page 178.

`-t`
`--tolerance`
(`=FLT`) The tolerance to switch from Monte Carlo integration to the central pixel value, see Section 8.1.1.5 [Sampling from a function], page 166.

`-p`
`--tunitinp`
The truncation column of the catalog is in units of pixels. By default, the truncation column is considered to be in units of the radial parameters of the profile (`--rcol`). Read it as 't-unit-in-p' for 'truncation unit in pixels'.

`-X`
`--xshift` (`=INT`) Shift all the profiles and enlarge the image along the first FITS axis, see n in Section 8.1.2 [If convolving afterwards], page 167. This is useful when you want to convolve the image afterwards. If you are using an external PSF, be sure to oversample it to the same scale used for creating the mock images. If a background image is specified, any possible value to this option is ignored.

`-Y`
`--yshift` (`=INT`) Similar to `--xshift` for the second FITS axis.

`-c`
`--prepforconv`
Shift all the profiles and enlarge the image based on half the width of the first Moffat or Gaussian profile in the catalog, considering any possible oversampling see Section 8.1.2 [If convolving afterwards], page 167. `--prepforconv` is only checked and possibly activated if `--xshift` and `--yshift` are both zero (after reading the command-line and configuration files). If a background image is specified, any possible value to this option is ignored.

`--magatpeak`
The magnitude column in the catalog (see Section 8.1.5.1 [MakeProfiles catalog], page 169) will be used to find the brightness only for the peak profile pixel, not

the full profile. Note that this is the flux of the profile's peak pixel in the final output of MakeProfiles. So beware of the oversampling, see Section 8.1.1.6 [Oversampling], page 167.

This option can be useful if you want to check a mock profile's total magnitude at various truncation radii. Without this option, no matter what the truncation radius is, the total magnitude will be the same as that given in the catalog. But with this option, the total magnitude will become brighter as you increase the truncation radius.

In sharper profiles, sometimes the accuracy of measuring the peak profile flux is more than the overall object brightness. In such cases, with this option, the final profile will be built such that its peak has the given magnitude, not the total profile.

CAUTION: If you want to use this option for comparing with observations, please note that MakeProfiles does not do convolution. Unless you have deconvolved your data, your images are convolved with the instrument and atmospheric PSF, see Section 8.1.1.2 [Point Spread Function], page 163. Particularly in sharper profiles, the flux in the peak pixel is strongly decreased after convolution. Also note that in such cases, besides de-convolution, you will have to set `--oversample=1` otherwise after resampling your profile with ImageWarp (see Section 6.4 [ImageWarp], page 109), the peak flux will be different.

`-R`
`--replace`

Do not add the pixels of each profile over the background (possibly crowded by other profiles), replace them. By default, when two profiles overlap, the final pixel value is the sum of all the profiles that overlap on that pixel. When this option is given, the pixels are not added but replaced by the newer profile's pixel and any value under it is lost.

When order matters, make sure to use this function with '`--numthreads=1`'. When multiple threads are used, the separate profiles are built asynchronously and not in order. Since order does not matter in an addition, this causes no problems by default but has to be considered when this option is given. Using multiple threads is no problem if the profiles are to be used as a mask with a blank or fixed value (see '`--mforflatpix`') since all their pixel values are the same.

Note that only non-zero pixels are replaced. With radial profiles (for example Sérsic or Moffat) only values above zero will be part of the profile. However, when using flat profiles with the '`--mforflatpix`' option, you should be careful not to give a `0.0` value as the flat profile's pixel value.

`-w`
`--circumwidth`

(`=FLT`) The width of the circumference if the profile is to be an elliptical circumference or annulus. See the explanations for this type of profile in `--fcol`.

`-z`
`--zeropoint`
(`=FLT`) The zero-point magnitude of the image.

Catalog: The value to all of these options is considered to be a column number, where counting starts from zero.

`--fcol` (`=INT`) The functional form of the profile with one of the values below. Note that this value will be converted to an integer before analysis using the internal type conversion of C. So for example 2.80 will be converted to 2.

- 0: Sérsic.
- 1: Moffat.
- 2: Gaussian.
- 3: Point source (a star).
- 4: Flat profile: all pixels have same value.
- 5: Circumference: same value for all pixels between the truncation radius (r_t) and $r_t - w$ where w is the value to the `--circumwidth`. Currently this is only intended to be used for making an elliptical annulus (with a width of 1 or 2 pixels).

`--xcol` (`=INT`) The center of the profiles along the first FITS axis (horizontal when viewed in SAO ds9). See the explanations for `--racol` for precedence when both image and WCS coordinate columns are given.

`--ycol` (`=INT`) The center of the profiles along the second FITS axis (vertical when viewed in SAO ds9). Similar to `--xcol`.

`--racol` (`=INT`) The profile center's right ascension. Along with `--deccol`, these WCS coordinate columns are not mandatory. If they are not given, the `--xcol` and `--ycol` options will be used to specify the profile's central position and vice-versa. However, if image coordinate columns (`--xcol` and `--ycol`) and WCS coordinate columns (`--racol` and `--deccol`) are given, the WCS coordinate columns take precedence and image coordinate columns will be ignored.

`--deccol` (`=INT`) The profile center's declination. Similar to `--racol`.

`--rcol` (`=INT`) The radius parameter of the profiles. Effective radius (r_e) if Sérsic, FWHM if Moffat or Gaussian.

`--ncol` (`=INT`) The Sérsic index (n) or Moffat β.

`--pcol` (`=INT`) The position angle (in degrees) of the profiles relative to the first FITS axis (horizontal when viewed in SAO ds9).

`--qcol` (`=INT`) The axis ratio of the profiles (minor axis divided by the major axis).

`--mcol` (`=INT`) The total pixelated magnitude of the profile within the truncation radius, see Section 8.1.4 [Profile magnitude], page 168.

`--tcol` (`=INT`) The truncation radius of this profile. By default it is in units of the radial parameter of the profile (the value in the `--rcol` of the catalog). If `--tunitinp` is given, this value is interpreted in units of pixels (prior to oversampling) irrespective of the profile.

`-F`
`--mforflatpix`
When making fixed value profiles (flat and circumference, see '`--fcol`'), don't use the value in the column specified by '`--mcol`' as the magnitude. Instead use it as the exact value that all the pixels of these profiles should have. This option is irrelevant for other types of profiles. This option is very useful for creating masks, or labeled regions in an image. Any integer, or floating point value can used in this column with this option, including `NaN` (or '`nan`', or '`NAN`', case is irrelevant), and infinities (`inf`, `-inf`, or `+inf`).

For example, with this option if you set the value in the magnitude column (`--mcol`) to `NaN`, you can create an elliptical or circular mask over an image (which can be given as the argument), see Section 6.1.3 [Blank pixels], page 77. Another useful application of this option is to create labeled elliptical or circular apertures in an image. To do this, set the value in the magnitude column to the label you want for this profile. This labeled image can then be used in combination with NoiseChisel's output (see Section 7.2.1.2 [NoiseChisel output], page 144) to do aperture photometry with MakeCatalog (see Section 7.3 [MakeCatalog], page 145).

Alternatively, if you want to mark regions of the image (for example with an elliptical circumference) and you don't want to use NaN values (as explained above) for some technical reason, you can get the minimum or maximum value in the image[9] using Arithmetic (see Section 6.2 [Arithmetic], page 83), then use that value in the magnitude column along with this option for all the profiles.

Please note that when using MakeProfiles on an already existing image, you have to set '`--oversample=1`'. Otherwise all the profiles will be scaled up based on the oversampling scale in your configuration files (see Section 4.2 [Configuration files], page 51) unless you have accounted for oversampling in your catalog.

WCS:

`--crpix1` (`=FLT`) The pixel coordinates of the WCS reference point on the first (horizontal) FITS axis (counting from 1).

`--crpix2` (`=FLT`) The pixel coordinates of the WCS reference point on the second (vertical) FITS axis (counting from 1).

`--crval1` (`=FLT`) The Right Ascension (RA) of the reference point.

`--crval2` (`=FLT`) The Declination of the reference point.

`--resolution`
(`=FLT`) The resolution of the non-oversampled image in units of arcseconds/pixel.

[9] The minimum will give a better result, because the maximum can be too high compared to most pixels in the image, making it harder to display.

8.1.5.3 MakeProfiles output

Besides the final merged image of all the profiles or individual profiles that can be built based on the input options, MakeProfiles will also create a log file in the current directory (where you run MockProfiles). The values for each column are explained in the first few commented (starting with `#` character). The log file includes the following information:

- The total magnitude of the profile in the image. This will be different from your input magnitude if the profile was not completely in the image.
- The number of pixels (in the oversampled image) which used Monte Carlo integration and not the central pixel value.
- The fraction of flux in the Monte Carlo integrated pixels.
- If an individual image was created or not.

8.2 MakeNoise

Real data are always buried in noise, therefore to finalize a simulation of real data (for example to test our observational algorithms) it is essential to add noise to the mock profiles created with MakeProfiles, see Section 8.1 [MakeProfiles], page 162. Below, the general principles and concepts to help understand how noise is quantified is discussed. MakeNoise options and argument are then discussed in Section 8.2.2 [Invoking MakeNoise], page 180.

8.2.1 Noise basics

Deep astronomical images, like those used in extragalactic studies seriously suffer from noise in the data. Generally speaking, the sources of noise in an astronomical image are photon counting noise and Instrumental noise which are discussed in detail below. We finish with a short introduction on how random numbers are generated and how you can determine the random number generator and seed value.

8.2.1.1 Photon counting noise

Thanks to the very accurate electronics used in today's detectors, this type of noise is the main cause of concern for extra galactic studies. It can generally be associate with the counting error that is known to have a Poisson distribution. The Poisson distribution is about counting. But counting is a discrete operation with only positive values, for example we can't count 3.2 or -2 of anything. We only count 0, 1, 2, 3 and so on. Therefore the Poisson distribution is also a discrete distribution, only applying to whole positive integers.

Let's assume the mean value of counting something is known. In this case, the number of electrons that are produced by photons in the CCD. Let's call this mean λ. Let's take k to represent the result of counting in one particular time we attempt to count. The probability density function of k can be written as:

$$f(k) = \frac{\lambda^k}{k!}e^{-\lambda}, \quad k \in \{0, 1, 2, 3, \ldots\}$$

Because the Poisson distribution is only applicable to positive values, it is by nature very skewed when λ is near zero. One qualitative way to imagine it is that there simply aren't enough integers smaller than λ, than there are larger integers. Therefore to accommodate all possibilities, it has to be skewed when λ is small.

But as λ becomes larger and larger, the distribution becomes more and more symmetric. One very useful property of the Poisson distribution is that the mean value is also its variance. When λ is very large, say $\lambda > 1000$, then the normal (Gaussian) distribution, see Section 8.1.1.2 [Point Spread Function], page 163, is an excellent approximation of the Poisson distribution with mean $\mu = \lambda$ and standard deviation $\sigma = \sqrt{\lambda}$.

We see that the variance or dispersion of the distribution depends on the mean value, and when it is large it can be approximated with a Gaussian that only has one free parameter ($\mu = \lambda$ and $\sigma = \sqrt{\lambda}$) instead of two that it originally has.

The astronomical objects after convolution with the PSF of the instrument, lie above a certain background flux. This background flux is defined to be the average flux of a region in the image that has absolutely no objects. The physical origin of this background value is the brightness of the atmosphere or possible stray light within the imaging instrument. It is thus an ideal definition, because in practice, what lies deep in the noise far lower than the detection limit is never known[10]. However, in a real image, a relatively large number of very faint objects can been fully buried in the noise. These undetected objects will bias the background measurement to slightly larger values. The sky value is therefore defined to be the average of the undetected regions in the image, so in an ideal case where all the objects have been detected, the sky value and background value are the same.

As longer wavelengths are used, the background value becomes more significant and also varies over a wide image field. Such variations are not currently implemented in MakeProfiles, but will be in the future. In a mock image, we have the luxury of setting the background value.

In each pixel of the canvas of pixels, the flux is the sum of contributions from various sources after convolution. Let's name this flux of the convolved sum of possibly overlapping objects, I_{nn}. nn representing 'no noise'. For now, let's assume the background is constant and represented by B. In practice the background values are larger than $\sim 1,000$ counts. Then the flux after adding noise is a random value taken from a Gaussian distribution with the following mean (μ) and standard deviation (σ):

$$\mu = B + I_{nn}, \quad \sigma = \sqrt{B + I_{nn}}$$

Since this type of noise is inherent in the objects we study, it is usually measured on the same scale as the astronomical objects, namely the magnitude system, see Section 8.1.3 [Flux Brightness and magnitude], page 167. It is then internally converted to the flux scale for further processing.

8.2.1.2 Instrumental noise

While taking images with a camera, a dark current is fed to the pixels, the variation of the value of this dark current over the pixels, also adds to the final image noise. Another source of noise is the readout noise that is produced by the electronics in the CCD that attempt to digitize the voltage produced by the photo-electrons in the analog to digital converter. In deep extra-galactic studies these sources of noise are not as significant as the noise of the background sky. Let C represent the combined standard deviation of all these sources

[10] See the section on sky in Akhlaghi M., Ichikawa. T. 2015. Astrophysical Journal Supplement Series.

of noise. If only this source of noise is present, the noised pixel value would be a random value chosen from a Gaussian distribution with

$$\mu = I_{nn}, \quad \sigma = \sqrt{C^2 + I_{nn}}$$

This type of noise is completely independent of the type of objects being studied, it is completely determined by the instrument. So the flux scale (and not magnitude scale) is most commonly used for this type of noise. In practice, this value is usually reported in ADUs not flux or electron counts. The gain value of the device can be used to convert between these two, see Section 8.1.3 [Flux Brightness and magnitude], page 167.

8.2.1.3 Final noised pixel value

Depending on the values you specify for B and C from the above, the final noised value for each pixel is a random value chosen from a Gaussian distribution with

$$\mu = B + I_{nn}, \quad \sigma = \sqrt{C^2 + B + I_{nn}}$$

8.2.1.4 Generating random numbers

As discussed above, to generate noise we need to make random samples of a particular distribution. So it is important to understand some general concepts regarding the generation of random numbers. For a very complete and nice introduction we strongly advise reading Donald Knuth's "The art of computer programming", volume 2, chapter 3[11]. Quoting from the GNU Scientific Library manual, "If you don't own it, you should stop reading right now, run to the nearest bookstore, and buy it"[12]!

Using only software, we can only produce what is called a psudo-random sequence of numbers. A true random number generator is a hardware (let's assume we have made sure it has no systematic biases), for example throwing dice or flipping coins (which have remained from the ancient times). More modern hardware methods use atmospheric noise, thermal noise or other types of external electromagnetic or quantum phenomena. All psudo-random number generators (software) require a seed to be the basis of the generation. The advantage of having a seed is that if you specify the same seed for multiple runs, you will get an identical sequence of random numbers which allows you to reproduce the same final noised image.

The programs in GNU Astronomy Utilities (for example MakeNoise or MakeProfiles) use the GNU Scientific Library (GSL) to generate random numbers. GSL allows the user to set the random number generator through environment variables, see Section 3.3.1.2 [Installation directory], page 36, for an introduction to environment variables. In the chapter titled "Random Number Generation" they have fully explained the various random number generators that are available (there are a lot of them!). Through the two environment variables `GSL_RNG_TYPE` and `GSL_RNG_SEED` you can specify the generator and its seed respectively.

[11] Knuth, Donald. 1998. The art of computer programming. Addison–Wesley. ISBN 0-201-89684-2

[12] For students, running to the library might be more affordable!

If you don't specify a value for `GSL_RNG_TYPE`, GSL will use its default random number generator type. The default type is sufficient for most general applications. If no value is given for the `GSL_RNG_SEED` environment variable and you have asked Gnuastro to read the seed from the environment (through the `--envseed` option), then GSL will use the default value of each generator to give identical outputs. If you don't explicitly tell Gnuastro programs to read the seed value from the environment variable, then they will use the system time (accurate to within a microsecond) to generate (apparently random) seeds. In this manner, every time you run the program, you will get a different random number distribution.

There are two ways you can specify values for these environment variables. You can call them on the same command-line for example:

```
$ GSL_RNG_TYPE="taus" GSL_RNG_SEED=345 astmknoise input.fits
```

In this manner the values will only be used for this particular execution of MakeNoise. To define them for the full period of your terminal session or script length, you can use the shell's `export` command (for a script remove the `$` signs):

```
$ export GSL_RNG_TYPE="taus"
$ export GSL_RNG_SEED=345
```

The subsequent programs which use GSL's random number generators will hence forth use these values in this session of the terminal you are running or while executing this script. In case you want set fixed values for these variables every time you use the GSL random number generator, you can add these two lines to your `.bashrc` startup script[13], see Section 3.3.1.2 [Installation directory], page 36.

[13] Don't forget that if you are going to give your scripts (that use the GSL random number generator) to others you have to make sure you also tell them to set these environment variable separately. So for scripts, it is best to keep all such variable definitions within the script, even if they are within your `.bashrc`.

NOTE: If the two environment variables `GSL_RNG_TYPE` and `GSL_RNG_SEED` are defined, GSL will report them by default, even if you don't use the `--envseed` option. For example you can see the top few lines of the output of MakeProfiles:

```
$ export GSL_RNG_TYPE="taus"
$ export GSL_RNG_SEED=345
$ astmkprof catalog.txt --envseed
GSL_RNG_TYPE=taus
GSL_RNG_SEED=345
MakeProfiles started on AAA BBB DD EE:FF:GG HHH
  - 6 profiles read from catalog.txt 0.000236 seconds
  - Random number generator (RNG) type: taus
  - RNG seed for all profiles: 345
```

The first two output lines (showing the names of the environment variables) are printed by GSL before MakeProfiles actually starts generating random numbers. The Gnuastro programs will report the values they use independently, you should check them for the final values used. For example if `--envseed` is not given, `GSL_RNG_SEED` will not be used and the last line shown above will not be printed. In the case of MakeProfiles, each profile will get its own seed value.

8.2.2 Invoking MakeNoise

MakeNoise will add noise to an existing image. The executable name is `astmknoise` with the following general template

```
$ astmknoise [OPTION ...] InputImage.fits
```

One line examples:

```
$ astmknoise --background=1000 --stdadd=20 mockimage.fits
```

If actual processing is to be done, the input image is a mandatory argument. The full list of options common to all the programs in Gnuastro can be seen in Section 4.1.4 [Common options], page 48. The output will have the same type as the input image, however the internal processing is done on a double precision floating point format. If the input values were integer types, then each floating point number will be rounded to the nearest integer away from zero. This might cause integer overflow if types with small ranges are used (for example images with a `BITPIX` of `8` which can only keep 256 values). This can be disabled with the `doubletype` option. The header of the output FITS file keeps all the parameters that were influential in making it. This is done for future reproducibility.

`-b`
`--background`
: (`=FLT`) The background pixel value for the image in units of magnitudes, see Section 8.2.1.1 [Photon counting noise], page 176, and Section 8.1.3 [Flux Brightness and magnitude], page 167.

`-z`
`--zeropoint`
: (`=FLT`) The zeropoint magnitude used to convert the value of `--background` (in units of magnitude) to flux, see Section 8.1.3 [Flux Brightness and magnitude], page 167.

`-s`
`--stdadd` (`=FLT`) The instrumental noise which is in units of flux, see Section 8.2.1.2 [Instrumental noise], page 177.

`-e`
`--envseed`
Use the `GSL_RNG_SEED` environment variable for the seed used in the random number generator, see Section 8.2.1.4 [Generating random numbers], page 178. With this option, the output image noise is always going to be identical (or reproducible).

`-d`
`--doubletype`
Save the output in the double precision floating point format that was used internally. This option will be most useful if the input images were of integer types.

9 High-level calculations

After the reduction of raw data (for example with the programs in Chapter 6 [Image manipulation], page 75) you will have reduced images/data ready for processing/analyzing (for example with the programs in Chapter 7 [Image analysis], page 128). But the processed/analyzed data (or catalogs) are still not enough to derive any scientific result. Even higher-level analysis is still needed to convert the observed magnitudes, sizes or volumes into physical quantities that we associate with each catalog entry or detected object which is the purpose of the tools in this section.

9.1 CosmicCalculator

To derive higher-level information regarding our sources in extra-galactic astronomy, cosmological calculations are necessary. In Gnuastro, CosmicCalculator is in charge of such calculations. Before discussing how CosmicCalculator is called and operates (in Section 9.1.3 [Invoking CosmicCalculator], page 187), it is important to provide a rough but mostly self sufficient review of the basics and the equations used in the analysis. In Section 9.1.1 [Distance on a 2D curved space], page 182, the basic idea of understanding distances in a curved and expanding 2D universe (which we can visualize) are reviewed. Having solidified the concepts there, in Section 9.1.2 [Extending distance concepts to 3D], page 186, the formalism is extended to the 3D universe we are trying to study in our research.

The focus here is obtaining a physical insight into these equations (mainly for the use in real observational studies). There are many books thoroughly deriving and proving all the equations with all possible initial conditions and assumptions for any abstract universe, interested readers can study those books.

9.1.1 Distance on a 2D curved space

The observations to date (for example the Plank 2013 results), have not measured the presence of a significant curvature in the universe. However to be generic (and allow its measurement if it does in fact exist), it is very important to create a framework that allows curvature. As 3D beings, it is impossible for us to mentally create (visualize) a picture of the curvature of a 3D volume in a 4D space. Hence, here we will assume a 2D surface and discuss distances on that 2D surface when it is flat, or when the 2D surface is curved (in a 3D space). Once the concepts have been created/visualized here, in Section 9.1.2 [Extending distance concepts to 3D], page 186, we will extend them to the real 3D universe we live in and hope to study.

To be more understandable (actively discuss from an observer's point of view) let's assume we have an imaginary 2D friend living on the 2D space (which *might* be curved in 3D). So here we will be working with it in its efforts to analyze distances on its 2D universe. The start of the analysis might seem too mundane, but since it is impossible to imagine a 3D curved space, it is important to review all the very basic concepts thoroughly for an easy transition to a universe we cannot visualize any more (a curved 3D space in 4D).

To start, let's assume a static (not expanding or shrinking), flat 2D surface similar to Figure 9.1 and that our 2D friend is observing its universe from point A. One of the most basic ways to parametrize this space is through the Cartesian coordinates (x, y). In Figure 9.1, the basic axises of these two coordinates are plotted. An infinitesimal change in

the direction of each axis is written as dx and dy. For each point, the infinitesimal changes are parallel with the respective axises and are not shown for clarity. Another very useful way of parametrizing this space is through polar coordinates. For each point, we define a radius (r) and angle (ϕ) from a fixed (but arbitrary) reference axis. In Figure 9.1 the infinitesimal changes for each polar coordinate are plotted for a random point and a dashed circle is shown for all points with the same radius.

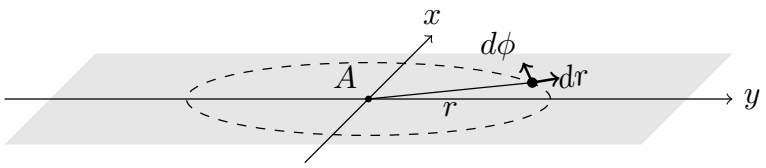


Figure 9.1: Two dimensional Cartesian and polar coordinates on a flat plane.

Assuming a certain position, which can be parametrized as (x, y), or (r, ϕ), a general infinitesimal change change in its position will place it in the coordinates $(x+dx, y+dy)$ and $(r + dr, \phi + d\phi)$. The distance (on the flat 2D surface) that is covered by this infinitesimal change in the static universe (ds_s, the subscript signifies the static nature of this universe) can be written as

$$ds_s = dx^2 + dy^2 = dr^2 + r^2 d\phi^2$$

The main question is this: how can our 2D friend incorporate the (possible) curvature in its universe when it is calculating distances? The universe it lives in might equally be a locally flat but globally curved surface like Figure 9.2. The answer to this question but for a 3D being (us) is the whole purpose to this discussion. So here we want to give our 2D friend (and later, ourselves) the tools to measure distances if the space (that hosts the objects) is curved.

Figure 9.2 assumes a spherical shell with radius R as the curved 2D plane for simplicity. The spherical shell is tangent to the 2D plane and only touches it at A. The result will be generalized afterwards. The first step in measuring the distance in a curved space is to imagine a third dimension along the z axis as shown in Figure 9.2. For simplicity, the z axis is assumed to pass through the center of the spherical shell. Our imaginary 2D friend cannot visualize the third dimension or a curved 2D surface within it, so the remainder of this discussion is purely abstract for it (similar to us being unable to visualize a 3D curved space in 4D). But since we are 3D creatures, we have the advantage of visualizing the following steps. Fortunately our 2D friend knows our mathematics, so it can follow along with us.

With the third axis added, a generic infinitesimal change over *the full* 3D space corresponds to the distance:

$$ds_s^2 = dx^2 + dy^2 + dz^2 = dr^2 + r^2 d\phi^2 + dz^2.$$

It is very important to recognize that this change of distance is for *any* point in the 3D space, not just those changes that occur on the 2D spherical shell of Figure 9.2. Recall that our 2D friend can only do measurements in the 2D spherical shell, not the full 3D space. So we have to constrain this general change to any change on the 2D spherical shell. To

do that, let's look at the arbitrary point P on the 2D spherical shell. Its image (P') on the flat plain is also displayed. From the dark triangle, we see that

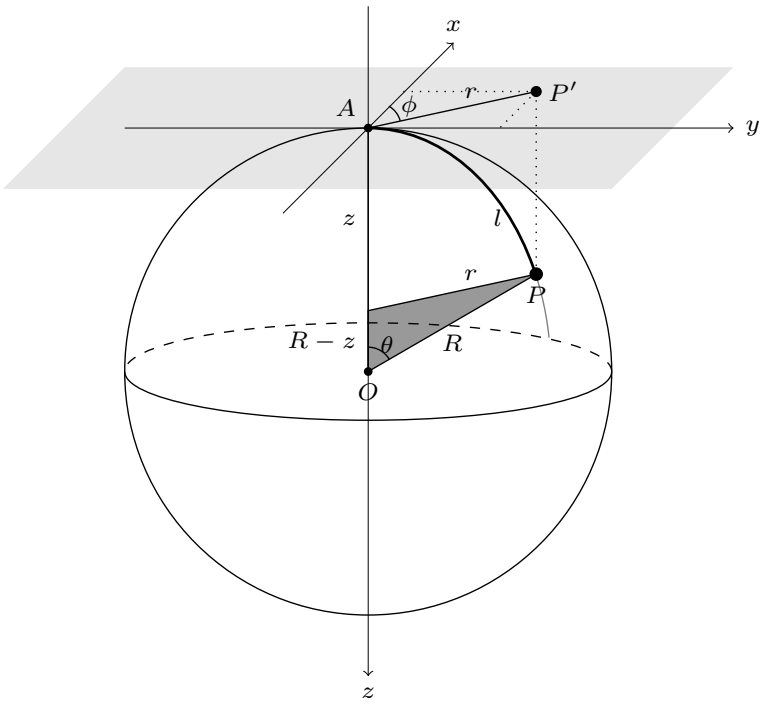


Figure 9.2: 2D spherical plane (centered on O) and flat plane (gray) tangent to it at point A.

$$\sin\theta = \frac{r}{R}, \quad \cos\theta = \frac{R-z}{R}.$$

These relations allow our 2D friend to find the value of z (an abstract dimension for it) as a function of r (distance on a flat 2D plane, which it can visualize) and thus eliminate z. From $\sin^2\theta + \cos^2\theta = 1$, we get $z^2 - 2Rz + r^2 = 0$ and solving for z, we find:

$$z = R\left(1 \pm \sqrt{1 - \frac{r^2}{R^2}}\right).$$

The $\pm$ can be understood from Figure 9.2: For each r, there are two points on the sphere, one in the upper hemisphere and one in the lower hemisphere. An infinitesimal change in r, will create the following infinitesimal change in z:

$$dz = \frac{\mp r}{R}\left(\frac{1}{\sqrt{1 - r^2/R^2}}\right)dr.$$

Using the positive signed equation instead of dz in the ds_s^2 equation above, we get:

$$ds_s^2 = \frac{dr^2}{1 - r^2/R^2} + r^2 d\phi^2.$$

The derivation above was done for a spherical shell of radius R as a curved 2D surface. To generalize it to any surface, we can define $K = 1/R^2$ as the curvature parameter. Then the general infinitesimal change in a static universe can be written as:

$$ds_s^2 = \frac{dr^2}{1 - Kr^2} + r^2 d\phi^2.$$

Therefore, we see that a positive K represents a real R which signifies a closed 2D spherical shell like Figure 9.2. When $K = 0$, we have a flat plane (Figure 9.1) and a negative K will correspond to an imaginary R. The latter two cases are open universes (where r can extend to infinity). However, when $K > 0$, we have a closed universe, where r cannot become larger than R as in Figure 9.2.

A very important issue that can be discussed now (while we are still in 2D and can actually visualize things) is that $\overrightarrow{r}$ is tangent to the curved space at the observer's position. In other words, it is on the gray flat surface of Figure 9.2, even when the universe if curved: $\overrightarrow{r} = P' - A$. Therefore for the point P on a curved space, the raw coordinate r is the distance to P', not P. The distance to the point P (at a specific coordinate r on the flat plane) on the curved surface (thick line in Figure 9.2) is called the *proper distance* and is displayed with l. For the specific example of Figure 9.2, the proper distance can be calculated with: $l = R\theta$ (θ is in radians). using the $\sin\theta$ relation found above, we can find l as a function of r:

$$\theta = \sin^{-1}\left(\frac{r}{R}\right) \quad \rightarrow \quad l(r) = R\sin^{-1}\left(\frac{r}{R}\right)$$

R is just an arbitrary constant and can be directly found from K, so for cleaner equations, it is common practice to set $R = 1$, which gives: $l(r) = \sin^{-1} r$. Also note that if $R = 1$, then $l = \theta$. Generally, depending on the the curvature, in a *static* universe the proper distance can be written as a function of the coordinate r as (from now on we are assuming $R = 1$):

$$l(r) = \sin^{-1}(r) \quad (K > 0), \qquad l(r) = r \quad (K = 0), \qquad l(r) = \sinh^{-1}(r) \quad (K < 0).$$

With l, the infinitesimal change of distance can be written in a more simpler and abstract form of

$$ds_s^2 = dl^2 + r^2 d\phi^2.$$

Until now, we had assumed a static universe (not changing with time). But our observations so far appear to indicate that the universe is expanding (isn't static). Since there is no reason to expect the observed expansion is unique to our particular position of the universe, we expect the universe to be expanding at all points with the same rate at the same time. Therefore, to add a time dependence to our distance measurements, we can simply add a multiplicative scaling factor, which is a function of time: $a(t)$. The functional form of $a(t)$ comes from the cosmology and the physics we assume for it: general relativity.

With this scaling factor, the proper distance will also depend on time. As the universe expands (moves), the distance will also move to larger values. We thus define a distance measure, or coordinate, that is independent of time and thus doesn't 'move' which we call the *comoving distance* and display with χ such that: $l(r,t) = \chi(r)a(t)$. We thus shift the r dependence of the proper distance we derived above for a static universe to the comoving distance:

$$\chi(r) = \sin^{-1}(r) \quad (K > 0), \qquad \chi(r) = r \quad (K = 0), \qquad \chi(r) = \sinh^{-1}(r) \quad (K < 0).$$

Therefore $\chi(r)$ is the proper distance of an object at a specific reference time: $t = t_r$ (the r subscript signifies "reference") when $a(t_r) = 1$. At any arbitrary moment ($t \neq t_r$) before or after t_r, the proper distance to the object can simply be scaled with $a(t)$. Measuring the change of distance in a time-dependent (expanding) universe will also involve the speed of the object changing positions. Hence, let's assume that we are only thinking about the change in distance caused by something (light) moving at the speed of light. This speed is postulated as the only constant and frame-of-reference-independent speed in the universe, making our calculations easier, light is also the major source of information we receive from the universe, so this is a reasonable assumption for most extra-galactic studies. We can thus parametrize the change in distance as

$$ds^2 = c^2dt^2 - a^2(t)ds_s^2 = c^2dt^2 - a^2(t)(d\chi^2 + r^2d\phi^2).$$

9.1.2 Extending distance concepts to 3D

The concepts of Section 9.1.1 [Distance on a 2D curved space], page 182, are here extended to a 3D space that *might* be curved in a 4D space. We can start with the generic infinitesimal distance in a static 3D universe, but this time not in spherical coordinates instead of polar coordinates. θ is shown in Figure 9.2, but here we are 3D beings, positioned on O (the center of the sphere) and the point O is tangent to a 4D-sphere. In our 3D space, a generic infinitesimal displacement will have the distance:

$$ds_s^2 = dx^2 + dy^2 + dz^2 = dr^2 + r^2(d\theta^2 + \sin^2\theta d\phi^2).$$

Like our 2D friend before, we now have to assume an abstract dimension which we cannot visualize. Let's call the fourth dimension w, then the general change in coordinates in the *full* four dimensional space will be:

$$ds_s^2 = dr^2 + r^2(d\theta^2 + \sin^2\theta d\phi^2) + dw^2.$$

But we can only work on a 3D curved space, so following exactly the same steps and conventions as our 2D friend, we arrive at:

$$ds_s^2 = \frac{dr^2}{1 - Kr^2} + r^2(d\theta^2 + \sin^2\theta d\phi^2).$$

In a non-static universe (with a scale factor a(t), the distance can be written as:

$$ds^2 = c^2dt^2 - a^2(t)[d\chi^2 + r^2(d\theta^2 + \sin^2\theta d\phi^2)].$$

9.1.3 Invoking CosmicCalculator

CosmicCalculator will calculate cosmological variables based on the input parameters. The executable name is `astcosmiccal` with the following general template

```
$ astcosmiccal [OPTION...] ...
```

One line examples:

```
$ astcosmiccal --onlyvolume -z=0.8
$ astcosmiccal -l=0.6964 -m=0.3036 -z=2.1
$ astcosmiccal --olambda=0.6964 --omatter=0.3036 --redshift=2.1
```

The input parameters can be given as command-line options or in the configuration files, see Section 4.2 [Configuration files], page 51. For a definition of the different parameters, please see the sections prior to this. By default, all the cosmological calculations will be printed in the standard output (the command-line mainly) along with a short description and units.

The options starting with `--only` will only do that single desired calculation and only print the final number (in the same units as reported by default). These options are very useful when you want to call CosmicCalculator from a script. The resulting number can simply be put into a shell variable (for example `vol`) with the following line, which will allow you to use the value for any other subsequent operation.

```
z=3.12
vol=$(astcosmiccal --redshift=$z --onlyvolume)
```

In a script, this operation might be necessary for a very large number of objects (thousands of galaxies in a catalog for example). So the fact that all the other default calculations are ignored will also help you get to your result faster. If you just want to inspect the value of a variable, the description (which comes with units) might be more useful. In that case, the following command might be better. The other parameters will also be calculated, but they are so fast that you will not notice on modern computers.

```
$ astcosmiccal --redshift=0.832 | grep volume
```

The full list of options is shown and described below:

`-z`
`--redshift`
(`=FLT`) The redshift of interest.

`-H`
`--H0` (`=FLT`) Current expansion rate (in km $\sec^{-1}$ Mpc^{-1}).

`-l`
`--olambda`
(`=FLT`) Cosmological constant density divided by the critical density in the current Universe ($\Omega_{\Lambda,0}$).

`-m`
`--omatter`
(`=FLT`) Matter (including massive neutrinos) density divided by the critical density in the current Universe ($\Omega_{m,0}$).

`-r`
`--oradiation`
(`=FLT`) Radiation density divided by the critical density in the current Universe ($\Omega_{r,0}$).

`-v`
`--onlyvolume`
Only print the comoving volume (in units of Mpc^3) until the desired redshift based on the input parameters. See explanations above for more on these types of options and how to effectively use them.

`-d`
`--onlyabsmagconv`
Only print the conversion factor for apparent magnitude to absolute magnitude. Note that this is practically the distance modulus added with $-2.5\log(1+z)$ for the the desired redshift based on the input parameters. See explanations above for more on these types of options and how to effectively use them.

10 Libraries

Each program in Gnuastro that was discussed in the prior chapters (or any program in general) is a collection of functions that is compiled into one executable file which can communicate directly with the outside world. The outside world in this context is the operating system. By communication, we mean that control is directly passed to a program from the operating system with a (possible) set of inputs and after it is finished, the program will pass control back to the operating system. For programs written in C and C++, the unique `main` function is in charge of this communication.

Similar to a program, a library is also a collection of functions that is compiled into one executable file. However, unlike programs, libraries don't have a `main` function. Therefore they can't communicate directly with the outside world. This gives you the chance to write your own `main` function and call library functions from within it. After compiling your program into a binary executable, you just have to *link* it to the library and you are ready to run (execute) your program. In this way, you can use Gnuastro at a much lower-level, and in combination with other libraries on your system, you can significantly boost your creativity.

This chapter starts with a basic introduction to libraries and how you can use them in Section 10.1 [Review of library fundamentals], page 189. The separate functions in the Gnuastro library are then introduced (classified by context) in Section 10.2 [Gnuastro library], page 197. If you end up rutinely using a fixed set of library functions, with a well-defined input and output, it will be much more beneficial if you define a program for the job. Therefore, in its Section 3.2.2 [Version controlled source], page 31, Gnuastro comes with the Section 11.6.2 [The TEMPLATE program], page 246, to easily define your own programs(s).

10.1 Review of library fundamentals

Gnuastro's libraries are written in the C programming language. In Section 11.1 [Why C programming language?], page 236, we have thoroughly discussed the reasons behind this choice. C was actually created to write Unix, thus understanding the way C works can greatly help in effectively using programs and libraries in all Unix-like operating systems. Therefore, in the following subsections some important aspects of C, as it relates to libraries (and thus programs that depend on them) on Unix are reviewed. First we will discuss header files in Section 10.1.1 [Headers], page 190, and then go onto Section 10.1.2 [Linking], page 193. This section finishes with Section 10.1.3 [Summary and example on libraries], page 195. If you are already familiar with these concepts, please skip this section and go directly to Section 10.2 [Gnuastro library], page 197.

In theory, a full operating system (or any software) can be written as one function. Such a software would not need any headers or linking (that are discussed in the subsections below). However, writing that single function and maintaining it (adding new features, fixing bugs, documentation and etc) would be a programmer or scientist's worst nightmare! Futhermore, all the hard work that went into creating it cannot be reused in other software: every other programmer or scientist would have to re-invent the wheel. The ultimate purpose behind libraries (which come with headers and have to be linked) is to address this problem and increase modularity: "the degree to which a system's components may be separated and

recombined" (from Wikipedia). The more modular the source code of a program or library, the easier maintaining it will be, and all the hard work that went into creating it can be reused for a wider range of problems.

10.1.1 Headers

C source code is read from top to bottom in the source file, therefore program components (for example variables, data structures and functions) should all be *defined* or *declared* closer to the top of the source file: before they are used. *Defining* something in C or C++ is jargon for providing its full details. *Declaraing* it, on the other-hand, is jargon for only providing the minimum information needed for the compiler to pass it temporarily and fill in the detailed definition later.

For a function, the *declaration* only contains the inputs and their data-types along with the output's type[1]. The *definition* adds to the declaration by including the exact details of what operations are done to the inputs to generate the output. As an example, take this simple summation function:

```
double
sum(double a, double b)
{
  return a + b;
}
```

What you see above is the *definition* of this function: it shows you (and the compiler) exactly what it does to the two `double` type inputs and that the output also has a `double` type. Note that a function's internal operations are rarely so simple and short, it can be arbitrarily long and complicated. This unreasonably short and simple function was chosen here for ease of reading. The declaration for this function is:

```
double
sum(double a, double b);
```

You can think of a function's declaration as a building's address in the city, and the definition as the building's complete blueprints. When the compiler confronts a call to a function during its processing, it doesn't need to know anything about how the inputs are processed to generate the output. Just as the postman doesn't need to know the inner structure of a building when delivering the mail. The declaration (address) is enough. Therefore by *declaring* the functions once at the start of the source files, we don't have to worry about *defining* them after they are used.

Even for a simple real-world operation (not a simple summation like above!), you will soon need many functions (for example, some for reading/preparing the inputs, some for the processing, and some for preparing the output). Although it is technically possible, managing all the necessary functions in one file is not easy and is contrary to the modularity principle (see Section 10.1 [Review of library fundamentals], page 189), for example the functions for preparing the input can be usable in your other projects with a different processing. Therefore, as we will see later (in Section 10.1.2 [Linking], page 193), the functions don't necessarily need to be defined in the source file where they are used. As long as their definitions are ultimately linked to the final executable, everything will be fine. For now, it is just important to remember that the functions that are called within

[1] Recall that in C, functions only have one output.

one source file must be declared within the source file (declarations are mandatory), but not necessarily defined there.

In the spirit of modularity, it is common to define contextually similar functions in one source file. For example, in Gnuastro, functions that calculate the median, mean and other statistical functions are defined in `lib/statistics.c`, while functions that deal directly with FITS files are defined in `lib/fits.c`.

Keeping the definition of similar functions in a separate file greatly helps their management and modularity, but this fact alone doesn't make things much easier for the caller's source code: recall that while definitions are optional, declarations are mandatory. So if this was all, the caller would have to manually copy and paste (*include*) all the declarations from the various source files into the file they are working on now. To address this problem, programmers have adopted the header file convention: the header file of a source code contains all the declarations that a caller would need to be able to use any of its functions. For example, in Gnuastro, `lib/statistics.c` (file containing function definitions) comes with `lib/gnuastro/statistics.h` (only containing function declarations).

The discussion above was mainly focused on functions, however, there are many more programming constructs such as pre-processor macros and data structures. Like functions, they also need to be known to the compiler when it confronts a call to them. So the header file also contains their definitions or declarations when they are necessary for the functions.

Pre-processor macros (or macros for short) are replaced with their defined value by the pre-processor before compilation. Conventionally they are written only in capital letters to be easily recognized. It is just important to understand that the compiler doesn't see the macros, it sees their fixed values. So when a header specifies macros you can do your programming without worrying about the actual values. The standard C types (for example `int`, or `float`) are very low-level and basic. We can collect multiple C types into a *structure* for a higher-level way to keep and pass-along data. See Section 10.2.4.2 [FITS macros and data structures], page 203, for some examples of macros and data structures.

The contents in the header need to be *include*d into the caller's source code with a special pre-processor command: `#include <path/to/header.h>`. As the name suggests, the *pre-processor* goes through the source code prior to the processor (or compiler). One of its jobs is to include, or merge, the contents of files that are mentioned with this directive in the source code. Therefore the compiler sees a single entity containing the contents of the main file and all the included files. This allows you to include many (sometimes thousands of) declarations into your code with only one line. Since the headers are also installed with the library into your system, you don't even need to keep a copy of them for each separate program, making things even more convenient.

Try opening some of the `.c` files in Gnuastro's `lib/` directory with a text editor to check out the include directives at the start of the file (after the copyright notice). Let's take `lib/fits.c` as an example. You will notice that Gnuastro's header files (like `gnuastro/fits.h`) are indeed within this directory (the `fits.h` file is in the `gnuastro/` directory). You will notice that files like `stdio.h`, or `string.h` are not in this directory (or anywhere within Gnuastro).

On most systems the basic C header files (like `stdio.h` and `string.h` mentioned above) are located in `/usr/include/`[2]. Your compiler is configured to automatically search that directory (and possibly others), so you don't have to explictly mention these directories. Go ahead, look into the `/usr/include` directory and find `stdio.h` for example. When the necessary header files are not in those specific libraries, the pre-processor can also search in places other than the current directory. You can specify those directories with this pre-processor option[3]:

`-I DIR` "Add the directory `DIR` to the list of directories to be searched for header files. Directories named by '-I' are searched before the standard system include directories. If the directory `DIR` is a standard system include directory, the option is ignored to ensure that the default search order for system directories and the special treatment of system headers are not defeated..." (quoted from the GNU Compiler Collection manual). Note that the space between `I` and the directory is optional and commonly not used.

If the pre-processor can't find the included files, it will abort with an error. Infact a common error when building programs that depend on a library is that the compiler doesn't not know where a library's header is (see Section 3.3.4 [Known issues], page 43). So you have to manually tell the compiler where to look for the library's headers with the `-I` option. For a small software with one or two source files, this can be done manually (see Section 10.1.3 [Summary and example on libraries], page 195). However, to enhance modularity, Gnuastro (and most other bin/libraries) contain many source files, so the compiler is invoked many times[4]. This makes manual addition or modification of this option practically impossible.

To solve this problem, in the GNU build system, there are conventional environment variables for the various kinds of compiler options (or flags). These environment variables are used in every call to the compiler (they can be empty). The environment variable used for the C Pre-Processor (or CPP) is `CPPFLAGS`. By giving `CPPFLAGS` a value once, you can be sure that each call to the compiler will be affected. See Section 3.3.4 [Known issues], page 43, for an example of how to set this variable at configure time.

As described in Section 3.3.1.2 [Installation directory], page 36, you can select the top installation directory of a software using the GNU build system, when you `./configure` it. All the separate components will be put in their separate subdirectory under that, for example the programs, compiled libraries and library headers will go into `$prefix/bin` (replace `$prefix` with a directory), `$prefix/lib`, and `$prefix/include` respectively. For enhanced modularity, libraries that contain diverse collections of functions (like GSL, WCSLIB, and Gnuastro), put their header files in a subdirectory unique to themselves. For example all Gnuastro's header files are installed in `$prefix/include/gnuastro`. In your source code, you need to keep the library's subdirectory when including the headers from such libraries, for example `#include <gnuastro/fits.h>`[5]. Not all libraries need to follow this convension, for example CFITSIO only has one header (`fitsio.h`) which is directly installed in `$prefix/include`.

[2] The `include/` directory name is taken from the pre-processor's `#include` directive, which is also the motivation behind the 'I' in the `-I` option to the pre-processor.

[3] Try running Gnuastro's `make` and find the directories given to the compiler with the `-I` option.

[4] Nearly every command you see being executed after running `make` is one call to the compiler.

[5] the top `$prefix/include` directory is usually known to the compiler

10.1.2 Linking

To enhance modularity, similar functions are defined in one source file (with a `.c` suffix, see Section 10.1.1 [Headers], page 190, for more). After running `make`, each human-readable, `.c` file is translated (or compiled) into a computer-readable "object" file (ending with `.o`). Note that object files are also created when building programs, they aren't particular to libraries. Try opening Gnuastro's `lib/` and `bin/progname/` directories after running `make` to see these object files[6]. Afterwards, the object files are *linked* together to create an executable program or a library.

The object files contain the full definition of the functions in the respective `.c` file along with a list of any other function (or generally "symbol") that is referenced there. To get a list of those functions you can use the `nm` program which is part of GNU Binutils. For example from the top Gnuastro directory, run:

```
$ nm bin/arithmetic/arithmetic.o
```

This will print a list of all the functions (more generally, 'symbols') that were called within `bin/arithmetic/arithmetic.c` along with some further information (for example a `T` in the second column shows that this function is actually defined here, `U` says that it is undefined here). Try opening the `.c` file to check some of these functions for your self. Run `info nm` for more information.

To recap, the *compiler* created the separate object files mentioned above for each `.c` file. The *linker* will then combine all the symbols of the various object files (and libraries) into one program or library. In the case of Arithmetic (a program) the contents of the object files in `bin/arithmetic/` are copied (and re-ordered) into one final executable file which we can run from the operating system. When the symbols (computer-readable function definitions in most cases) are copied into the output like this, we call the process *static* linking. Let's have a closer look at static linking: we'll assume you have installed Gnuastro into the default `/usr/local/` directory (see Section 3.3.1.2 [Installation directory], page 36). If you tried the `nm` command on one of Arithmetic's object files above, then with the command below you can confirm that all the functions that were defined in the object files (had a `T` in the second column) are also defined in the `astarithmetic` executable:

```
$ nm /usr/local/bin/astarithmetic
```

But you will notice that there are still many undefined symbols (have a `U` in the second column). One class of such functions are Gnuastro's own library functions that start with '`gal_`':

```
$ nm /usr/local/bin/astarithmetic | grep gal_
```

These undefined symbols (functions) will be linked to the executable everytime you run arithmetic. Therefore they are known as dynamically *linked* libraries[7]. When the functions of a library need to be dynamically linked, the library is known as a shared library. As we saw above, static linking is done when the executable is being built. However, when a library is linked dynamically, its symbols are only checked with the available libraries at build time: they are not actually copied into the executable. Everytime you run the program, the linker

[6] Gnuastro uses GNU Libtool for portable library creation. Libtool will also make a `.lo` file for each `.c` file when building libraries (`.lo` files are human-readable).

[7] Do not confuse dynamicly *linked* libraries with dynamically *loaded* libraries. The former (that is discussed here) are only loaded once at the program startup. However, the latter can be loaded anytime during the program's execution, they are also known as plugins.

will be activated and will try to link the program to the installed library before it starts. If you want all the libraries to be statically linked to the executables, you have to tell Libtool (which Gnuastro uses for the linking) to disable shared libraries at configure time[8]:

```
$ configure --disable-shared
```

Try configuring, statically building and installing Gnuastro with the command above. Then check the `gal_` symbols in the installed Arithmetic executable like before. You will see that they are actually copied this time (have a `T` in the second column). If the second column doesn't convince you, look at the executable file size with the following command:

```
$ ls -lh /usr/local/bin/astarithmetic
```

It should be around 4.2 Megabytes with this static linking. If you configure and build Gnuastro again with shared libraries enabled (which is the default), you will notice that it is roughly 100 Kilobytes! This huge difference would have been very significant in the old days, but with the roughly Terabyte storages commonly in use today, it is negligible. Fortunately, output file size is not the only benefit of dynamic linking: since it links to the libraries at run-time (rather than build-time), you don't have to re-build a higher-level program or library when an update comes for one of the lower-level libraries it depends on. You just install the new low-level library and it will automatically be used next time in your higher-level tools. To be fair, this also creates a few complications[9]:

- Reproducibility: Even though your high-level tool has the same version as before, with the updated library, you might not get the same results.
- Broken links: if some functions have been changed or removed in the updated library, then the linker will abort with an error at run-time. Therefore you need to re-build your higher-level program or library.

To see a list of all the shared libraries that are needed for a program or a shared library to run, you can use the GNU C library's `ldd`[10] program, for example:

```
$ ldd /usr/local/bin/astarithmetic
```

Library file names start with a `lib` and end with suffix depending on their type as described below. In between these two is the name of the library, for example `libgnuastro.a` (Gnuastro's static library) and `libgsl.so.0.0.0` (GSL's shared library).

- A static library is known as an archive file and has a `.a` suffix. A static library is not an executable file.
- A shared library ends with a `.so.X.Y.Z` suffix and is executable. The three numbers in the prefix are the version of the shared library. Shared library versions are defined to allow multiple versions of a shared library simultaneously on a system and to help detect possible updates in the library and programs that depend on it by the linker. It is very important to mention that this version number is differnent from from the software version number (see Section 1.5 [Version numbering], page 5), so do not confuse the two. See the "Library interface versions" chapter of GNU Libtool for more.

[8] Libtool is very common and is commonly used. Therefore, you can use this option to configure on most programs using the GNU build system if you want static linking.

[9] Both of these can be avoided by joining the mailing lists of the lower-level libraries and checking the changes in newer versions before installing them. Updates that result in such behaviors are generally heavily emphasized in the release notes.

[10] If your operating system is not using the GNU C library, you might need another tool.

For each shared library, we also have two symbolic links ending with `.so.X` and `.so`. They are automatically set by the installer, but you can change them (point them to another version of the library) when you have multiple versions on your system.

For those libraries that use GNU Libtool (including Gnuastro and its dependencies), both static and dynamic libraries are built and installed in the `prefix/lib/` directory (see Section 3.3.1.2 [Installation directory], page 36). In this way other programs can make which ever kind of link that they want.

To link with a library, the linker needs to know where to find the library. You do that with two separate options to the linker (see Section 10.1.3 [Summary and example on libraries], page 195, for an example):

`-L DIR` Will tell the linker to look into `DIR` for the libraries. For example `-L/usr/local/lib`, or `-L/home/yourname/.local/lib`. You can make multiple calls to this option, so the linker looks into several directories. Note that the space between `L` and the directory is optional and commonly not used.

`-lLIBRARY`

Specify the unique name of a library to be linked. As discussed above, library file names have fixed parts which must not be given to this option. So `-lgsl` will guide the linker to either look for `libgsl.a` or `libgsl.so` (depending on the type of linking it is suppose to do). You can link many libraries by repeated calls to this option.

Very important: The place of this option on the command line matters. This is often a source of confusion for beginners, so let's assume you have asked the linker to link with library A using this option. As soon as the linker confronts this option, it looks into the list of the undefined symbols it has found until that point and does a search in library A for any of those symbols. If any pending undefined symbol is found in library A, it is used. After the search in undefined symbols is complete, the contents of library A are completely discarded from the linker's memory. Therefore, if a later object file or library uses an unlinked symbol in library A, the linker will abort after it has finished its search in all the input libraries or object files.

As an example, Gnuastro's `gal_array_dlog10_array` function depends on the `log10` function of the C Math library (specified with `-lm`). So the proper way to link something that uses this function is `-lgnuastro -lm`. If instead, you give: `-lm -lgnuastro` the linker will complain and abort.

10.1.3 Summary and example on libraries

After the mostly abstract discussions of Section 10.1.1 [Headers], page 190, and Section 10.1.2 [Linking], page 193, we'll give a small tutorial here. But before that, let's recall the general steps of how your source code is prepared, compiled and linked to the librays it depends on so you can run it:

1. The **pre-processor** includes the header (`.h`) files into the function definition (`.c`) files, expands pre-processor macros and generally prepares the human-readable source for compilation (reviewed in Section 10.1.1 [Headers], page 190).

2. The **compiler** will translate (compile) the human-readable contents of each source (merged `.c` and the `.h` files, or generally the output of the pre-processor) into the computer-readable code of `.o` files.
3. The **linker** will link the called function definitions from various compiled files to create one unified object. When the unified product has a `main` function, this function is the product's only entry point, enabling the operating system or user to directly interact with it, so the product is a program. When the product doesn't have a `main` function, the linker's product is a library and its exported functions can be linked to other executables (it has many entry points).

The GNU Compiler Collection (or GCC for short) will do all three steps. So as a first example, from Gnuastro's source, go to `tests/lib/`. This directory contains the library tests, you can use these as some simple tutorials. For this demonstration, we will compile and run the `arraymanip.c`. This small program will call Gnuastro library for some simple operations on an array (open it and have a look). To compile this program, run this command inside the directory containing it.

```
$ gcc arraymanip.c -lgnuastro -lm -o arraymanip
```

The two `-lgnuastro` and `-lm` options (in this order) tell GCC to first link with the Gnuastro library and then with C's math library. The `-o` option is used to specify the name of the output executable, without it the output file name will be `a.out` (on most OSs), independent of your input file name(s).

If your top Gnuastro installation directory (let's call it `$prefix`, see Section 3.3.1.2 [Installation directory], page 36) is not recognized by GCC, you will get pre-processor errors for unknown header files. Once you fix it, you will get linker errors for undefined functions. To fix both, you should run GCC as follows: additionally telling it which directories it can find Gnuastro's headers and compiled library (see Section 10.1.1 [Headers], page 190, and Section 10.1.2 [Linking], page 193):

```
$ gcc -I$prefix/include -L$prefix/lib arraymanip.c -lgnuastro -lm     \
      -o arraymanip
```

This single command has done all the preprocessor, compilation and linker operations. Therefore no intermediate files (object files in particular) were created, only a single output executable was created. You are now ready to run the program with:

```
$ ./arraymanip
```

The Gnuastro functions called by this program only needed to be linked with the C math library. But if your program needs WCS coordinate transformations, needs to read a FITS file, needs special math operations (which include its linear algebra operations), or you want it to run on multiple CPU threads, you also need to add these libraries in the call to GCC: `-lgnuastro -lwcs -lcfitsio -lgsl -lgslcblas -pthread -lm`. In Section 10.2 [Gnuastro library], page 197, where each function is documented, it is mentioned which libraries (if any) must also be linked when you call a function. If you feel all these linkings can be confusing, please have a look at Section 10.1.4 [Automatic linking script], page 196.

10.1.4 Automatic linking script

The number and order of libraries that are necessary for linking a program with Gnuastro library might be too confusing when you need to compile a small program for one particular job (with one source file). To solve this problem, please add these lines to your

`~/.bashrc` with a text editor and replace `PREFIX` with Gnuastro's installation directory (see Section 3.3.1.2 [Installation directory], page 36).

```
# Compile and correctly link Gnuastro related programs
function gnuastro-gcc {
  libtool --mode=link gcc $1.c $2 PREFIX/lib/libgnuastro.la -o $1
}
```

This tiny shell function uses GNU Libtool to find the necessary libraries to link against. Libtool uses the same libraries that were checked when you configured Gnuastro to link your program. So in the future, if Gnuastro's prerequisite libraries change, you don't have to worry. Afterwards, when you want to compile a C program that uses Gnuastro library (for example `myprog.c`), just run the following command from the directory which contains the source file:

```
gnuastro-gcc myprog
```

If your program needs to link with libraries other than those needed by Gnuastro, you can specify them as a second argument (within double quotes):

```
gnuastro-gcc myprog "-lfirst -lsecond"
```

If you also want to execute the compiled program without typing a second command, add `&& ./myprog` to either of the lines above, like the example below. If your program can take arguments, then you can also include them afterwards.

```
gnuastro-gcc myprog && ./myprog
```

10.2 Gnuastro library

Gnuastro library's exported constructs (functions, macros, data structures, or global variables) are classified by context into multiple header files (see Section 10.1.1 [Headers], page 190)[11]. In this section, the functions in each header will be discussed under a separate sub-section, which includes the name of the header. Assuming a function declaration is in `headername.h`, you can include its declaration in your source code with:

```
# include <gnuastro/headername.h>
```

The names of all constructs are prefixed with `gal_headername_` (or `GAL_HEADERNAME_` for macros). The `gal_` prefix stands for *G*NU *A*stronomy *L*ibrary. The `headername` prefix is replaced by the header file name (without the `.h`) which contains the construct.

All Gnuastro library functions are compiled into a single file which can be linked on the command-line with the `-lgnuastro` option, (see Section 10.1.2 [Linking], page 193, and Section 10.1.3 [Summary and example on libraries], page 195, for an introduction). Gnuastro library is a high-level library which depends on lower level libraries for some operations (see Section 3.1 [Dependencies], page 24). Therefore if at least one of Gnuastro's functions in your program use functions from the dependencies, you will also need to link those dependencies after linking with Gnuastro. The outside libraries that need to be linked for such functions are mentioned following the function name. See Section 10.1.4 [Automatic linking script], page 196, for a small shell function to worry about the libraries to link against and let you focus on your exciting science.

[11] Within Gnuastro's source, all installed `.h` files in `lib/gnuastro/` are accompanied by a `.c` file in `/lib/`.

Libraries are still under heavy development: Gnuastro was initially created to be a collection of command-line programs. However, as the programs and their the shared functions grew, internal (not installed) libraries were added. With the 0.2 release, the libraries are installable. Because of this history in these early phases, the libraries are not fully programmer friendly yet: they abort the program on an error, their naming and arguments are not fully uniform, or modular, and most of the interesting functions (that are currently only used within one program) are still internal to the program. During the next releases, Gnuastro's library structure will undergo significant evolution to address all these problems. It will stabilize with the removal of this notice. Check the `NEWS` file for interface changes.

10.2.1 Installation information (`gnuastro.h`)

The `gnuastro/gnuastro.h` header contains information about the full Gnuastro installation on your system. Gnuastro developers should note that this is the only header that is not available within Gnuastro, it is only available to a Gnuastro library user *after* installation. Within Gnuastro, `config.h` (which is included in every Gnuastro `.c` file, see Section 11.5 [Coding conventions], page 240) should have more than enough information about the overall Gnuastro installation.

`GAL_GNUASTRO_VERSION` [Macro]

This macro can be used as a string literal[12] containing the version of Gnuastro that is being used. See Section 1.5 [Version numbering], page 5, for the version formats. For example:

```
printf("Gnuastro version: %s\n", GAL_GNUASTRO_VERSION);
```

or

```
char *gnuastro_version=GAL_GNUASTRO_VERSION;
```

`GAL_GNUASTRO_PTHREAD_BARRIER` [Macro]

The POSIX threads standard define barriers as an optional requirement. Therefore, some operating systems choose to not include it. While installing Gnuastro, we check if your system has this construct and if so, we give this macro a value of `1`, otherwise it will have a value of `0`. see Section 10.2.11.1 [Implementation of `pthread_barrier`], page 231, for more.

10.2.2 Array manipulation (`array.h`)

When working on arrays, certain operations are commonly necessary. It is very easy to write a loop for such operations, however such loops can unnecessarily make the code harder to write, read, and debug. So, it is much more cleaner if these operations are defined elsewhere. The declarations of these functions are available in `gnuastro/array.h`.

In C, an array is a contiguous region of memory. So for most operations here, the dimentionality of the data is irrelevant, we just need the total size of the array. Please note that these functions are still very rudimentary, primarily intended for Gnuastro's internal operations. In future releases, these functions will be updated to be more suitable for a generic library.

[12] `https://en.wikipedia.org/wiki/String_literal`

`void` [Function]
`gal_array_uchar_init_on_region` (*unsigned char* `*in`, *const unsigned char* `v`, *size_t* `start`, *size_t* `s0`, *size_t* `s1`, *size_t* `is1`)
Initialize a region of the array `in` to the value `v`. The region is defined by the 1D index of the first pixel (`start`) and its C array (opposite of FITS array) width (`s0`) and height (`s1`). The full C array width is defined by `is1`.

`void` [Function]
`gal_array_long_init` (*long* `*in`, *size_t* `size`, *const long* `v`)
Initialize the long type array `in` with `size` elements to the value `v`.

`void` [Function]
`gal_array_long_init_on_region` (*long* `*in`, *const long* `v`, *size_t* `start`, *size_t* `s0`, *size_t* `s1`, *size_t* `is1`)
Similar to `gal_array_uchar_init_on_region` but for long type arrays.

`void` [Function]
`gal_array_uchar_copy` (*unsigned char* `*in`, *size_t* `size`, *unsigned char* `**out`)
Allocate space for and copy the input array `in` with `size` elements into an output array (`out`).

`void` [Function]
`gal_array_float_copy` (*float* `*in`, *size_t* `size`, *float* `**out`)
Similar to `gal_array_uchar_copy` but for the single precision floating point type.

`void` [Function]
`gal_array_float_copy_noalloc` (*float* `*in`, *size_t* `size`, *float* `*out`)
Similar to `gal_array_float_copy` but the `out` array must already be allocated.

`void` [Function]
`gal_array_fset_const` (*float* `*in`, *size_t* `size`, *float* `a`)
Set the input array `in` with `size` elements to the constant value `a`.

`void` [Function]
`gal_array_freplace_value` (*float* `*in`, *size_t* `size`, *float* `from`, *float* `to`)
Replace all elements with value `from` to value `to` in the `in` array with `size` elements. `from` can be a NaN value, in this case, C's `isnan` function is used, not `==`.

`void` [Function]
`gal_array_freplace_nonnans` (*float* `*in`, *size_t* `size`, *float* `to`)
Replace any non-NaN value in the array `in` with `size` elements to the value `to`.

`void` [Function]
`gal_array_no_nans` (*float* `*in`, *size_t* `*size`)
Move all the non-NaN elements in the array `in` to the start of the array so that the non-NaN alements are contiguous. This is useful for cases where you want to sort the data. Note that before this function, `size` must point to the initial size of the array. After this function returns, size will point to the new size of the array, with only non-Nan elements.

`void` [Function]
`gal_array_no_nans_double` (*double* `*in`, *size_t* `*size`)
Similar to `gal_array_no_nans` but for double types.

`void` [Function]
`gal_array_uchar_replace` (*unsigned char* `*in`, *size_t* `size`, *unsigned char* `from`, *unsigned char* `to`)
Replace all elements with value `from` to value `to` in the `in` array with `size` elements.

`void` [Function]
`gal_array_fmultip_const` (*float* `*in`, *size_t* `size`, *float* `a`)
Multiply all elements in the float array `in` with size `size` by the constant `a`.

`void` [Function]
`gal_array_fsum_const` (*float* `*in`, *size_t* `size`, *float* `a`)
Add all elements in the float array `in` with size `size` by the constant `a`.

`float *` [Function]
`gal_array_fsum_arrays` (*float* `*in1`, *float* `*in2`, *size_t* `size`)
Add the two arrays `in1` and `in2` and keep the result in a newly allocated array. The returned value is a pointer to that array.

`void` [Function]
`gal_array_dmultip_const` (*double* `*in`, *size_t* `size`, *double* `a`)
Multiply the double type array `in` with `size` elements with the constant `a`.

`void` [Function]
`gal_array_dmultip_arrays` (*double* `*in1`, *double* `*in2`, *size_t* `size`)
Multiply the two double type arrays `in1` with `in2` (both with `size` elements) with each other and store the output in the first array.

`void` [Function]
`gal_array_ddivide_const` (*double* `*in`, *size_t* `size`, *double* `a`)
Divide the double type array `in` with `size` elements by the constant `a`.

`void` [Function]
`gal_array_dconst_divide` (*double* `*in`, *size_t* `size`, *double* `a`)
Divide the constant `a` by the array `in` with `size` and store the result for each pixel in the array.

`void` [Function]
`gal_array_ddivide_arrays` (*double* `*in1`, *double* `*in2`, *size_t* `size`)
Divide each element of the two double array `in1` by the elements in `in2` (both with `size` elements) and store the output in the first array.

`void` [Function]
`gal_array_dsum_const` (*double* `*in`, *size_t* `size`, *double* `a`)
Add the double type array `in` with `size` elements with the constant `a`.

`void` [Function]
`gal_array_dsum_arrays` (*double* `*in1`, *double* `*in2`, *size_t* `size`)
Add the two double type arrays `in1` with `in2` (both with `size` elements) with each other and store the output in the first array.

`void` [Function]
`gal_array_dsubtract_const` (*double* `*in`, *size_t* `size`, *double* `a`)
Subtract the double type array `in` with `size` elements from the constant `a`.

`void` [Function]
`gal_array_dconst_subtract` (*double* `*in`, *size_t* `size`, *double* `a`)
Subtract the constant `a` from the array `in` with `size` and store the result for each pixel in the array.

`void` [Function]
`gal_array_dsubtract_arrays` (*double* `*in1`, *double* `*in2`, *size_t* `size`)
Subtract each element of th the double type array `in2` from `in1` and put the result in the first array.

`void` [Function]
`gal_array_dpower_const` (*double* `*in`, *size_t* `size`, *double* `a`)
Each element in the intput array is taken to the power of of `a` and stored in place.

`void` [Function]
`gal_array_dconst_power` (*double* `*in`, *size_t* `size`, *double* `a`)
The constant `a` is taken to the power of each element in the intput array (`in`) and stored in place.

`void` [Function]
`gal_array_dpower_arrays` (*double* `*in1`, *double* `*in2`, *size_t* `size`)
Each element in the first input array is taken to the power of the same element in the second array.

`void` [Function]
`gal_array_dlog_array` (*double* `*in1`, *size_t* `size`)
The natural logarithm of the input array is taken and stored in place.

`void` [Function]
`gal_array_dlog10_array` (*double* `*in1`, *size_t* `size`)
The base-10 logarithm of the input array is taken and stored in place.

`void` [Function]
`gal_array_dabs_array` (*double* `*in1`, *size_t* `size`)
Replace each element of the input array with its absolute value.

10.2.3 Bounding box (`box.h`)

Functions related to reporing a the bouding box of certain inputs are declared in `gnuastro/box.h`. All coordinates in this header are in the FITS format (first axis is the horizontal and the second axis is vertical).

`void` [Function]
`gal_box_ellipse_in_box` (*double* `A`, *double* `B`, *double* `theta_rad`, *long* `*width`)
Any ellipse can be enclosed into a rectangular box. The purpose of this function is to give the height and width of that box. `A` is the ellipse major axis, `B` is the minor axis, `theta_rad` is the position angle in radians. The `width` array will contain the output size in long integer type. `width[0]`, and `width[1]` are the number of pixels along the first and second FITS axis.

`void` [Function]
`gal_box_border_from_center` (*double* `xc`, *double* `yc`, *long* `*width`, *long* `*fpixel`, *long* `*lpixel`)
Given the center (`xc` and `yc`) and width (two element array) of a box, return the coordinates of the first `fpixel` and last `lpixel` pixels (both are two element arrays.

`int` [Function]
`gal_box_overlap` (*long* `*naxes`, *long* `*fpixel_i`, *long* `*lpixel_i`, *long* `*fpixel_o`, *long* `*lpixel_o`)
We have an image of size `naxes` and want to get the overlap of the image with a box. This function will return 1 if there is an overlap and 0 if there isn't. The input and output box are specified by their first and last pixels. When there is an overlap, the coordinates of the first and last pixels of the overlap will be put in `fpixel_o` and `lpixel_o`.

10.2.4 FITS files (`fits.h`)

The FITS format is the most common format to store data (images and tables) in astronomy. The CFITSIO library already provides a very good low-level set of functions for manipulating FITS data. Some CFITSIO operations (with well-defined inputs and outputs) are commonly needed in a special order. Therefore Gnuastro comes with the macros, data structures and functions introduced in this section. These functions are actually wrappers around CFITSIO functions for a higher-level (easier) access to FITS data. So if you feel these functions don't exactly do what you want, we strongly recommend reading the CFITSIO manual to use it directly.

All the functions and macros introduced in this section are declared in `gnuastro/fits.h`. When you include this header, you are also including CFITSIO's `fitsio.h` header. So you don't need to explicitly include `fitsio.h` anymore and can freely use any of its macros or functions in your code along with those discussed here. In Section 10.2.4.1 [CFITSIO `datatype`], page 202, we first have a look at the way data types are referenced in CFITSIO (and thus Gnuastro). Section 10.2.4.2 [FITS macros and data structures], page 203, defines the macros and data structures that can be used in functions for a unified format. This subsection finished with Section 10.2.4.3 [FITS functions], page 205, which introduce the different Gnuastro functions available.

10.2.4.1 CFITSIO `datatype`

Data can have multiple types: standards to convert a number of bits (which can only have values of 0 and 1) to different number formats (for example integer or floating point numbers). To store data, the FITS standard defines two major formats: images and tables. An image has a single type for all its data elements (a pixel in a 2D image). A table

has a single type for every one of its columns. In both cases, the FITS header stores this information. All the FITS data types correspond to C types (for example `short`, `int`, `float`, or `long`).

In images, the `BITPIX` header keyword specifies the type of data in each pixel. Within CFITSIO, the different acceptable values of `BITPIX` are stored as macros ending with `_IMG`. For example `BYTE_IMG`, or `DOUBLE_IMG`. See Section 4.1, "CFITSIO Definitions", in the CFITSIO manual for the recognized CFITSIO types and their macros. However, the FITS standard also accepts tables: each column in a FITS table has a unique `TFORM` header keyword which specifies the type of data in that column. The variety of types in a FITS table can be much larger than an image and thus the standard for identifying them differs.

To address these different formats of identifying the type in a uniform infra-structure, CFITSIO identifies a general collection of macros referring to types which can be internally used in all cases (images and tables). Thus in CFITSIO, the variable name `datatype` when we are referring to this general collection of types. The image types are a subset of the table column types, so each acceptable `datatype` corresponds to one of the table types. The macros for each recognized CFITSIO `datatype` start with `T`, for example `TBYTE`, `TDOUBLE`. In CFITSIO (and thus here in Gnuastro), the `datatype` is most commonly used, so this variable only accepts one of the accepted `T` macros. When the `BITPIX` value is desired (as input or output), the variable is called `bitpix`.

In Gnuastro, the functions `gal_fits_bitpix_to_datatype` and `gal_fits_tform_to_datatype` are provided to convert `bitpix` and `tform` values to `datatype`.

CFITSIO integer types: CFITSIO does not use the standard fixed integer size types of `stdint.h`! It uses the subjective `short`, `int` and `long` variables which can differ in size from system to system. For example, the FITS standard defines `LONG_IMG` as a 32bit signed integer type, but CFITSIO converts it to a local `long` which is 64 bits on a modern (64 bit) machine. To use CFITSIO, we had to adopt the same convention in Gnuastro. This issue can cause confusions, so be careful!

10.2.4.2 FITS macros and data structures

To facilitate handling of fixed/standard values and store, or pass multiple related variables in your programs, Gnuastro defines the following FITS related macros and data structures, see Section 10.1.1 [Headers], page 190, for a more detailed discussion.

`GAL_FITS_STRING_BLANK` [Macro]
`GAL_FITS_BYTE_BLANK` [Macro]
`GAL_FITS_LOGICAL_BLANK` [Macro]
`GAL_FITS_SHORT_BLANK` [Macro]
`GAL_FITS_LONG_BLANK` [Macro]
`GAL_FITS_LLONG_BLANK` [Macro]
`GAL_FITS_FLOAT_BLANK` [Macro]
`GAL_FITS_DOUBLE_BLANK` [Macro]
`GAL_FITS_INT_BLANK` [Macro]
`GAL_FITS_SBYTE_BLANK` [Macro]
`GAL_FITS_UINT_BLANK` [Macro]
`GAL_FITS_USHORT_BLANK` [Macro]

`GAL_FITS_ULONG_BLANK` [Macro]

These macros keep Gnuastro's blank value constant for all the recognized datatypes in CFITSIO (see Section 6.1.3 [Blank pixels], page 77, for a discussion on blank values). The values for the different types are mostly the lowest or highest possible value for that type (for unsigned types and 8-bit types, the maximum is used). So you can safely ignore the actual value and simply use these values to check. See explanation under the Section 10.2.4.1 [CFITSIO `datatype`], page 202, function for more on types of data in FITS.

The `float` and `double` types use the IEEE NaN values and for string it is a `NULL` pointer. By definition, a NaN value fails in every condition check, so a NaN value is not equal to itself and the check `variable==GAL_FITS_FLOAT_BLANK` will fail even if the value in `variable` is NaN. Therefore, to check for blank values in floating point types be careful to use C's `isnan` function.

`gal_fits_key` [Structure]

Structure for reading FITS header keywords. This structure will keep the name and value of one keyword. The value will be stored in the element corresponding to the type of the value. When multiple keywords must be read, you can define an array of this structure, for example the following declaration will allocate an array of five keywords to be read.

```
struct gal_fits_key keys[5];
```

The names of the keywords can then be stored separately in the array and this expression will store the values in each element.

```
gal_fits_read_keywords("filename.fits", hdu, gal_fits_key, 5);
```

The FITS standard defines a maximum length for the value of keyword, so the space for the `str` element is statically allocated (`FLEN_VALUE`, which is defined in CFITSIO). So if the string value is necessary where `gal_fits_key` is no longer available, then you have to allocate space dynamically and copy the string there.

```
struct gal_fits_key
{
  int            status;        /* CFITSIO status.        */
  char         *keyname;        /* Name of keyword.       */
  int          datatype;        /* Type of keyword value. */
  char  str[FLEN_VALUE];        /* String value.          */
  unsigned char       u;        /* Byte value.            */
  short               s;        /* Short integer value.   */
  long                l;        /* Long integer value.    */
  LONGLONG            L;        /* Long Long value.       */
  float               f;        /* Float value.           */
  double              d;        /* Double value.          */
};
```

`gal_fits_key_ll` [Structure]

Structure for writing FITS keywords. This structure is used for one keyword. However, with the `next` element, it can link to another keyword thus creating a linked list to add any number of keywords easily and at any step during your program. Adding

new keywords doesn't have to be done manually, the `gal_fits_add_to_key_ll` and `gal_fits_add_to_key_ll_end` will add new keywords for you to the start or end of the list to make a last-in-first-out or first-in-last-out list. See Section 10.2.5 [Linked lists (`linkedlist.h`)], page 210, for more on the nature of linked lists and more similar structures.

```
struct gal_fits_key_ll
{
  int                    kfree;   /* ==1, free keyword name.   */
  int                    vfree;   /* ==1, free keyword value.  */
  int                    cfree;   /* ==1, free comment.        */
  int                 datatype;   /* Keyword value datatype.   */
  char                *keyname;   /* Keyword Name.             */
  void                  *value;   /* Keyword value.            */
  char                *comment;   /* Keyword comment.          */
  char                   *unit;   /* Keyword unit.             */
  struct gal_fits_key_ll *next;   /* Pointer next keyword.     */
};
```

10.2.4.3 FITS functions

Gnuastro provides the following functions to deal with FITS data related operations. FITS data can have a variety of types, see Section 10.2.4.1 [CFITSIO `datatype`], page 202, for a discussion on this, in particular the integer variables named `datatype`, `bitpix`, and `tform`. See Section 10.2.4.2 [FITS macros and data structures], page 203, for the strure and macro definitions.

`void` [Function]
`gal_fits_io_error` (*int* `status`, *char* `*message`)

Report the input or output error as a string and print it along with a short statement that an error in CFITSIO occurred. All CFITSIO functions will change the value of `status` if an error occurs. This function will print the CFITSIO error string from the value of `status` followed by an optional `message` that you can add to it. If the value of `message` is `NULL`, then a default message will be printed.

`int` [Function]
`gal_fits_name_is_fits` (*char* `*name`)

Return `1` if the string pointed by `name` is a standard FITS filename (or more generally: can be read by CFITSIO) and `0` if it isn't. See Section 4.1.2 [Arguments], page 46, for the standard FITS filenames.

`int` [Function]
`gal_fits_suffix_is_fits` (*char* `*suffix`)

Return `1` if the string pointed by `suffix` is a standard FITS suffix (or more generally: can be read by CFITSIO) and `0` if it isn't. `suffix` can contain a dot or not (for example both `.fits` and `fits` will return `1`. See Section 4.1.2 [Arguments], page 46, for the standard FITS filenames.

`int` [Function]
`gal_fits_bitpix_to_datatype` (*int* `bitpix`)
Return the datatype corresponding to the given `bitpix` (value to the `BITPIX` header keyword) in a FITS image.

`int` [Function]
`gal_fits_tform_to_datatype` (*char* `tform`)
Return the datatype corresponding to the given `tform` character of a FITS table column: value to the `TFORM` header keyword for the column. Note that in the FITS standard, `TFORM` values are characters.

`void` [Function]
`gal_fits_img_bitpix_size` (*fitsfile* `*fptr`, *int* `*bitpix`, *long* `*naxes`)
Return the datatype (in FITS `BITPIX` format) and image size of a FITS HDU specified with the `fitsfile` pointer (defined in CFITSIO). So the HDU must have been already opened. If the number of dimentions is not 2, this function will retun an error and abort.

`void *` [Function]
`gal_fits_datatype_blank` (*int* `datatype`)
Allocate the necessary space and put the blank value of type `datatype` in it. Finally, return the pointer to the allocated space. This pointer is commonly necessary when calling CFITSIO read functions as `nulval`. See Section 10.2.4.1 [CFITSIO `datatype`], page 202.

`void *` [Function]
`gal_fits_datatype_alloc` (*size_t* `size`, *int* `datatype`)
Allocate an array of `size` elements of type `datatype`. See Section 10.2.4.1 [CFITSIO `datatype`], page 202.

`void` [Function]
`gal_fits_blank_to_value` (*void* `*array`, *int* `datatype`, *size_t* `size`, *void* `*value`)
Convert the blank values in `array` (with `size` elements) into the value pointed by `value`.

`void` [Function]
`gal_fits_change_type` (*void* `*in`, *int* `inbitpix`, *size_t* `size`, *int* `anyblank`, *void* `**out`, *int* `outbitpix`)
Convert the `in` array (with `inbitpix` type) into an array of type `outbitpix` that is pointed by `out`. Space will be allocated for the output array within this function.

`void` [Function]
`gal_fits_num_hdus` (*char* `*filename`, *int* `*numhdu`)
Find the number of HDUs in `filename` and store it in the space pointed to by `numhdu`.

`void` [Function]
`gal_fits_read_hdu` (*char* `*filename`, *char* `*hdu`, *unsigned char* `img0_tab1`, *fitsfile* `**outfptr`)
Open the HDU `hdu` (a string) of the `filename` FITS file into the `outfptr`. See `--hdu` option in Section 4.1.4.1 [Input/Output options], page 48, for the accepted HDU formats.

`void` [Function]
`gal_fits_read_keywords` (*char* `*filename`, *char* `*hdu`, *struct gal_fits_key* `*keys`, *size_t* `num`)

Read `num` keywords (identified in the `keys` array) in the `hdu` HDU of `filename`. See Section 10.2.4.2 [FITS macros and data structures], page 203, for the definition of `gal_fits_key` and its recommended usage with this function.

CFITSIO will look for the next keyword starting from the last keyword found. Therefore, it is most efficient to specify the keywords in the same order as they appear in the desired HDU.

`void` [Function]
`gal_fits_add_to_key_ll` (*struct gal_fits_key_ll* `**list`, *int* `datatype`, *char* `*keyname`, *int* `kfree`, *void* `*value`, *int* `vfree`, *char* `*comment`, *int* `cfree`, *char* `*unit`)

Add an element to the top of the `gal_fits_key_ll` linked list for writing keywords. This structure (or linked list) is used to write FITS keywords and is fully described in Section 10.2.4.2 [FITS macros and data structures], page 203. Each variable also has the same name as the structure element that is described there.

Assuming all the keywords were added with this function, you will have a last-in-first-out list: as the the name suggests, the last keyword you add to the list will be the first one that gets popped out and thus written in the FITS file header.

`void` [Function]
`gal_fits_add_to_key_ll_end` (*struct gal_fits_key_ll* `**list`, *int* `datatype`, *char* `*keyname`, *int* `kfree`, *void* `*value`, *int* `vfree`, *char* `*comment`, *int* `cfree`, *char* `*unit`)

Similar to `gal_fits_add_to_key_ll`, except the keyword is added to the end of the list, creating a first-in-first-out list. Therefore the order the keywords are written to the FITS header is the same as the order you have called this function.

`void` [Function]
`gal_fits_file_name_in_keywords` (*char* `*keynamebase`, *char* `*filename`, *struct gal_fits_key_ll* `**list`)

Put a filename into the `gal_fits_key_ll` list to later write into a HDU header. The FITS standard sets a maximum length for the value of a keyword. This creates problems with file names (including directories), because they can become very long. Therefore, when the filename is longer than the maximum length of a FITS keyword value, this function will break it into several keywords. The `keynamebase` string will be appended with a `_N` (N>0) and used as the keyword name.

`void` [Function]
`gal_fits_add_wcs_to_header` (*fitsfile* `*fptr`, *char* `*wcsheader`, *int* `nkeyrec`)

Add the WCS information into the header of the HDU pointed to by `fptr`. The WCS information must already be converted into a long string with the FITS conventions. To help in identifying the WCS information, a few blank lines and a title will be added ontop.

`void` [Function]
`gal_fits_update_keys` (*fitsfile* `*fptr`, *struct gal_fits_key_ll* `**keylist`)

Given the an opened FITS HDU (pointed to by `fptr`), this function will write, or update, all the keywords given in `keylist`. See Section 10.2.4.2 [FITS macros and data structures], page 203, for more information on the `gal_fits_key_ll` structure or linked list.

`void` [Function]
`gal_fits_write_keys_version` (*fitsfile* `*fptr`, *struct gal_fits_key_ll* `*headers`, *char* `*spack_string`)

Write or update (all the) keyword(s) in `headers` into the FITS pointer, but also the date, name of your program (`spack_string`), along with the verisons of Gnuastro, CFITSIO, WCSLIB (when available) into the header, see Section 4.7 [Output headers], page 61. Since the data processing depends on the versions of the libraries you have used, it is strongly recommended to include this information in every FITS output. See Section 10.2.4.2 [FITS macros and data structures], page 203, for more information on the `gal_fits_key_ll` structure or linked list.

`void` [Function]
`gal_fits_read_wcs_from_pointer` (*fitsfile* `*fptr`, *int* `*nwcs`, *struct wcsprm* `**wcs`, *size_t* `hstartwcs`, *size_t* `hendwcs`)

Read the WCSLIB WCS structure (`wcs`, and `nwcs` the number of WCS structures) from the FITS pointer `fptr`. In some cases, it might be necessary to only consider header keywords between a certain range. You can define the range of keywords to look for WCS information with the `hstartwcs` and `hendwcs` arguments. The range of header keywords will only be considered when the value of `hstartwcs` is less than `hendwcs`, so if you don't want to limit the range, set both values to 0. After you have finished using the WCS structure, use `wcsvfree(&nwcs,&wcs);` to free it.

WCSLIB is not thread-safe: WCSLIB's `wcspih` function is used by to read the FITS keywords into WCSLIB's internal WCS structures. Unfortunately this function is not thread-safe. Therefore, be sure to call that function (or Gnuastro's `gal_fits_read_wcs_from_pointer` and `gal_fits_read_wcs`) before spinning off threads, see Section 10.2.11 [Multithreaded programming (`threads.h`)], page 230.

`void` [Function]
`gal_fits_read_wcs` (*char* `*filename`, *char* `*hdu`, *size_t* `hstartwcs`, *size_t* `hendwcs`, *int* `*nwcs`, *struct wcsprm* `**wcs`)

Read the WCSLIB WCS structure (`wcs` and `nwcs` the number of WCS structures) from a FITS file (`hdu` in `filename`). This is basically just a wrapper around the `gal_fits_read_wcs_from_pointer` function, see that function for more.

`int` [Function]
`gal_fits_hdu_to_array` (*char* `*filename`, *char* `*hdu`, *int* `*bitpix`, *void* `**array`, *size_t* `*s0`, *size_t* `*s1`)

Read the FITS image (in the `hdu` extension of `filename`). The image array will be stored in `array` and the number of pixels in the first and second C axis will be

stored in `s0` and `s1`. The type of the array will be kept in `bitpix`. See `--hdu` in Section 4.1.4.1 [Input/Output options], page 48, for the acceptable formats of `hdu`.

`void` [Function]
`gal_fits_array_to_file` (*char* `*filename`, *char* `*extname`, *int* `bitpix`, *void* `*array`, *size_t* `s0`, *size_t* `s1`, *int* `anyblank`, *struct wcsprm* `*wcs`, *struct gal_fits_key_ll* `*headers`, *char* `*spack_string`)

Write the array `array` (with sizes `s0` and `s1`, type `bitpix`, and WCS structure `wcs`) into the FITS file `filename` and extension name `extname`. If the array has any blank pixels, then you can set `anyblank` to 1. All the headers in `headers` will also be written to this extension and `spack_string` is the name of your program (which can include a version number), which will be written in the output header.

`void` [Function]
`gal_fits_atof_correct_wcs` (*char* `*filename`, *char* `*hdu`, *int* `bitpix`, *void* `*array`, *size_t* `s0`, *size_t* `s1`, *char* `*wcsheader`, *int* `wcsnkeyrec`, *double* `*crpix`, *char* `*spack_string`)

This is essentially the same as `gal_fits_array_to_file`, but through the two element `crpix` array, you can change the final `CRPIX` keywords that will be written in the FITS file.

`void` [Function]
`gal_fits_table_size` (*fitsfile* `*fitsptr`, *size_t* `*nrows`, *size_t* `*ncols`)

Get the number of rows (`nrows`) and columns (`ncols`) in a FITS table within the `fitsptr`.

`int` [Function]
`gal_fits_table_type` (*fitsfile* `*fptr`)

Return the type of the table: `ASCII_TBL` and `BINARY_TBL` for an ASCII or binary table. These two macros are defined by CFITSIO.

`void` [Function]
`gal_fits_file_or_ext_name` (*char* `*inputname`, *char* `*inhdu`, *int* `othernameset`, *char* `**othername`, *char* `*ohdu`, *int* `ohduset`, *char* `*type`)

This is mainly a function used in Gnuastro programs. Since a FITS image can have many extensions, usually related data are stored within one FITS file. For example an image, its masked pixels, and the sky standard deviation of the noise are various extensions of one FITS file. It would be very inconvenient if the same file name would have to be repeated in a call to a program for all images that are within one file.

To easily manage things, within Gnuastro, each input variable also has an integer variable with the same name, but suffixed with `set`. The job of this function is to use the name of the main input image and based on the other arguments, determine what filename should be used for each type. The final argument (`type`) is the type of input you want, for example, if you were expecting a mask image, you can give the value `"mask"` as type. `type` is just used to report errors.

`void` [Function]
`gal_fits_file_to_float` (*char* `*inputname`, *char* `*maskname`, *char* `*inhdu`, *char* `*mhdu`, *float* `**img`, *int* `*inbitpix`, *int* `*anyblank`, *size_t* `*ins0`, *size_t* `*ins1`)

Read a FITS image HDU (`inhdu` in `inputname`) as a `float` type array. An optional mask image and mask hdu can also be given. Single precision floating point types are one of the most common types to store data: the observation noise is usually much larger than the extra precision (and thus volume in your computer, and slower processing) offered by double precision floating point types. With this function, so when you work internally with floating points, you can use this function to directly read a FITS image array into a floating point array.

`void` [Function]
`gal_fits_file_to_double` (*char* `*inputname`, *char* `*maskname`, *char* `*inhdu`, *char* `*mhdu`, *double* `**img`, *int* `*inbitpix`, *int* `*anyblank`, *size_t* `*ins0`, *size_t* `*ins1`)

Similar to `gal_fits_file_to_float`, but for double precision floating point type. In some contexts, the internal operations must be done in double precision floating point types, irrespective of the input type and this function can be used to directly read a file into an array of this type.

`void` [Function]
`gal_fits_file_to_long` (*char* `*inputname`, *char* `*inhdu`, *long* `**img`, *int* `*inbitpix`, *int* `*anyblank`, *size_t* `*ins0`, *size_t* `*ins1`)

Similar to `gal_fits_file_to_float`, but will read the image specified by `inputname` (`inhdu`) into a long integer type. Long integer types are commonly used to store things like segmentation maps or labeled images.

`void` [Function]
`gal_fits_prep_float_kernel` (*char* `*inputname`, *char* `*inhdu`, *float* `**outkernel`, *size_t* `*ins0`, *size_t* `*ins1`)

Read `inputname` (`inhdu`) will be read as a floating point array to be used as a convolution kernel. Convolution kernels are commonly necessary in astronomical image processing, but they need to have certain properties: their dimensions have to be an odd number and they must not have any blank pixels. In both cases, this function will abort with an error notice.

10.2.5 Linked lists (`linkedlist.h`)

An array is a contiguous region of memory that is very efficient and easy to use for recording and later accessing the data. One major problem with an array is that the number of elements that go into it must be known in advance. This is where linked lists can be very useful: like a metal chain, each *node* in a linked list is an independent C structure, keeping its own data along with pointer(s) to its immediate neighbor(s). Below we have one simple linked list node structure along with an ASCII art schematic of how we can use the `next` pointer to add any number of elements to the list that we want. The linked list is terminated when `next` is a `NULL` pointer.

```
struct float_ll          /*     ---------    ---------              */
```

```
{                         /*     | Value |    | Value |              */
  float          value;   /*     |  ---  |    |  ---  |              */
  struct ll_node *next;   /*     |  next-|--> |  next-|--> NULL      */
}                         /*     ---------    ---------              */
```

The schematic shows another great advantage of linked lists: it is very easy to add or remove a node anywhere in the list (you just have to change one pointer in the structure above). You initially define a variable of this type with a `NULL` pointer as shown below, then you use functions provided for that type of linked list to add elements to the list or pop elements from it.

```
struct float_ll *mylist=NULL
```

When you add an element to the list, it is conventionally added to the "top" of the list: the general list pointer will point to the newly created node, which will point to the previously created node and so on. So when you "pop" from the top of the list, you are actually retrieving the last value you put in and changing the list pointer to the next oldest node. This is thus known as a "last-in-first-out" list. This is the most convenient type of linked list (easier to implement and faster to process). Alternatively, you can add each newly created node at the end of the list. If you do that, you will get a "first-in-first-out" list. But that will force you to go through the whole list for each new element that is created (this can slow down the processing)[13].

Each node in a linked list doesn't have to have a single pointer to the next node like the example above. We can define each node with two pointers to both the next and previous neighbors, this is called a "Doubly linked list". In general, lists are very powerful and simple constructs that can be very useful. But going into more detail would be out of the scope of this short introduction in this book. Wikipedia (`https://en.wikipedia.org/wiki/Linked_list`) has a nice and more thorough discussion of the various types of lists, so for more, please have a look at that article.

In this section we will review the functions and structures that are available in Gnuastro for working on linked lists. For each linked-list node sturcture, we will first introduce the structure, then the functions for working on the structure. All these structures and functions are defined and declared in `gnuastro/linkedlist.h`.

`gal_linkedlist_fll` [Structure]

Float linked list (`fll`). Each node in this singly-linked list can have a single float type value.

```
struct gal_linkedlist_fll
{
    float                       v;
    struct gal_linkedlist_fll *next;
};
```

[13] A better way to get a first-in-first-out is to first keep the data as last-in-first-out until they are all read. Afterwards, pop each node and immediately add it to the new list: practically reversing the last-in-first-out list to a first-in-first-out one.

`void` [Function]
`gal_linkedlist_print_fll_array` (*struct gal_linkedlist_fll* `**afll`, *size_t* `num`)
Print all the values within the linked list to the output terminal. This can be used for inspecting the status of the list during development.

`void` [Function]
`gal_linkedlist_add_to_fll` (*struct gal_linkedlist_fll* `**list`, *float* `value`)
Allocate space for a new node in `list`, and put `value` into it. The new node will be added to the top of the list. So after this function, the already existing nodes will be appended to the new node and `list` will point to the newly created node.

`void` [Function]
`gal_linkedlist_pop_from_fll` (*struct gal_linkedlist_fll* `**list`, *float* `*value`)
Pop a node from the top of `list` and put its values in `value`. This function will also free the allocated space for the popped node and after this function, `list` will point to the next node.

`size_t` [Function]
`gal_linkedlist_num_in_fll` (*struct gal_linkedlist_fll* `*list`)
Return the number of nodes in `list`, the contents of `list` will be untouched.

`void` [Function]
`gal_linkedlist_fll_to_array` (*struct gal_linkedlist_fll* `*list`, *float* `**f`, *size_t* `*num`)
Put the elements of `list` into an array `f` and store the number of nodes in the list in `num`. The array will be filled in the same order as popped elements from the list. This function will allocate space for the `num` element `f` array internally, so you have to free it afterwards.

`void` [Function]
`gal_linkedlist_free_fll` (*struct gal_linkedlist_fll* `*list`)
Free the space for all the nodes in `list`. Note that any values within the nodes will also be discarded. This can be used when the list is no longer relevant for you. For example, you might have already converted the list to an array for future usage with `gal_linkedlist_fll_to_array`.

`void` [Function]
`gal_linkedlist_free_fll_array` (*struct gal_linkedlist_fll* `**afll`, *size_t* `num`)
Free an array of `gal_linkedlist_fll` linked lists (`num` elements). Each element of the array is a pointer to a linked list, so we first need to free the separate linked lists, then the array.

`gal_linkedlist_tdll` [Structure]
Two doubles linked list (`tdll`. Each node in this singly-linked list can have two double type values. It can be used to store an unknown number of floating point coordinates (for example RA and Dec).

```
struct gal_linkedlist_tdll
{
    double                          a;
```

```
    double                          b;
    struct gal_linkedlist_tdll *next;
};
```

`void` [Function]
`gal_linkedlist_add_to_tdll` (*struct gal_linkedlist_tdll* `**list`, *double* `a`, *double* `b`)

Allocate space for a new node in `list`, and put the values `a` and `b` into it. The new node will be added to the top of the list. So after this function, the already existing nodes will be appended to the new node and `list` will point to the newly created node.

`void` [Function]
`gal_linkedlist_pop_from_tdll` (*struct gal_linkedlist_tdll* `**list`, *double* `*a`, *double* `*b`)

Pop a node from the top of `list` and put its values in `a` and `b`. This function will also free the allocated space for the popped node and after this function, `list` will point to the next node.

`size_t` [Function]
`gal_linkedlist_num_int_dll` (*struct gal_linkedlist_tdll* `*list`)

Return the number of nodes in `list`. The contents of `list` will be untouched.

`void` [Function]
`gal_linkedlist_tdll_to_array_inv` (*struct gal_linkedlist_tdll* `*list`, *double* `**d`, *size_t* `*num`)

Put the elements of `list` into an array `d` in inverse order and store the number of nodes in the list in `num`. This function will allocate space for the 2×`num` element array internally so you have to free it afterwards. The two values for each node will be stored adjacently, so the array can be seen as a 2D array with `num` rows and 2 columns.

`void` [Function]
`gal_linkedlist_free_tdll` (*struct gal_linkedlist_tdll* `*list`)

Free the space for all the nodes in `list`. Note that any values within the nodes will also be discarded. This can be used when the list is no longer relevant for you. For example, you might have already converted the list to an array for future usage with `gal_linkedlist_tdll_to_array_inv`.

`gal_linkedlist_stll` [Structure]

String linked list (`stll`. Each node in this singly-linked list has a string variable.

```
struct gal_linkedlist_stll
{
    char                           *v;
    struct gal_linkedlist_stll *next;
};
```

`void` [Function]
`gal_linkedlist_add_to_stll` (*struct gal_linkedlist_stll* `**list`, *char* `*value`)

Allocate space for a new node in `list`, and store the `value` pointer into it. The new node will be added to the top of the list. So after this function, the already existing nodes will be appended to the new node and `list` will point to the newly created node.

It is important to remember that this function will only copy the pointer in the node, it will not allocate space for the string. So the string either has to be a literal string (with a fixed address) or dynamically allocated when this linked list is to be filled and used within one function.

`void` [Function]
`gal_linkedlist_pop_from_stll` (*struct gal_linkedlist_stll* `**list`, *char* `**value`)

Pop a node from the top of `list` and put its pointer in `value`. This function will also free the allocated space for the popped node and after this function, `list` will point to the next node.

`void` [Function]
`gal_linkedlist_reverse_stll` (*struct gal_linkedlist_stll* `**list`)

Reverse the linked list. The `gal_linkedlist_add_to_stll` function will make a last-in-first-out list, but commonly you will need a first-in-first-out list. This function can thus be useful after adding to the list has finished.

`void` [Function]
`gal_linkedlist_print_stll` (*struct gal_linkedlist_stll* `*list`)

Print all the values within the linked list to the output terminal. This can be used for inspecting the status of the list during development.

`size_t` [Function]
`gal_linkedlist_num_in_stll` (*struct gal_linkedlist_stll* `*list`)

Return the number of nodes in `list`. The contents of `list` will be untouched.

`gal_linkedlist_sll` [Structure]

`size_t` linked list (`sll`. Each node in this singly-linked list contains a `size_t` value. The `size_t` type is commonly used for "size" related values: it is an un-signed type (negative values are not defined for it), therefore it is ideal for dealing with things like pixel indexes in an image (which can not be negative and can be very large).

```
struct gal_linkedlist_sll
{
    size_t                      v;
    struct gal_linkedlist_sll *next;
};
```

`void` [Function]
`gal_linkedlist_add_to_sll` (*struct gal_linkedlist_sll* `**list`, *size_t* `value`)

Allocate space for a new node in `list`, and store `value` into it. The new node will be added to the top of the list. So after this function, the already existing nodes will be appended to the new node and `list` will point to the newly created node.

`void` [Function]
`gal_linkedlist_pop_from_sll` (*struct gal_linkedlist_sll* `**list`, *size_t* `*value`)
Pop a node from the top of `list` and put its value in `value`. This function will also free the allocated space for the popped node and after this function, `list` will point to the next node.

`size_t` [Function]
`gal_linkedlist_num_in_sll` (*struct gal_linkedlist_sll* `*list`)
Return the number of nodes in `list`. The contents of `list` will be untouched.

`void` [Function]
`gal_linkedlist_print_sll` (*struct gal_linkedlist_sll* `*list`)
Print all the values within the linked list to the output terminal. This can be used for inspecting the status of the list during development.

`void` [Function]
`gal_linkedlist_sll_to_array` (*struct gal_linkedlist_sll* `*list`, *size_t* `**s`, *size_t* `*num`, *int* `inverse`)
Put the elements of `list` into an array `s` and store the number of nodes in the list in `num`. This function will allocate space for the `num` element array internally so you have to free it afterwards. If the value in `inverse` is `1`, then the first popped value in the list will be the last element in the array, otherwise, the array will be filled in the same order that the elements were popped.

`void` [Function]
`gal_linkedlist_free_sll` (*struct gal_linkedlist_sll* `*list`)
Free the space for all the nodes in `list`. Note that any values within the nodes will also be discarded. This can be used when the list is no longer relevant for you. For example, you might have already converted the list to an array for future usage with `gal_linkedlist_sll_to_array`.

`gal_linkedlist_osll` [Structure]
Ordered `size_t` linked list (`osll`. Each node in this singly-linked list contains a `size_t` value and a floating point value. The floating point value is used as a reference to add new nodes in a sorted manner. At any moment, the first popped node in this list will have the smallest `tosort` value, and subsequent nodes will have larger to values.

```
struct gal_linkedlist_osll
{
  size_t                        v;
  float                         s;
  struct gal_linkedlist_osll *next;
};
```

`void` [Function]
`gal_linkedlist_add_to_osll` (*struct gal_linkedlist_osll* `**list`, *size_t* `value`, *float* `tosort`)
Allocate space for a new node in `list`, and store `value` into it. The new node will be added to the list based on the `tosort` value.

`void` [Function]
`gal_linkedlist_pop_from_osll` (*struct gal_linkedlist_osll* `**list`, *size_t* `*value`, *float* `*sortvalue`)

Pop a node from the top of `list`, put its value in `value` and its sort value in `sortvalue`. This function will also free the allocated space for the popped node and after this function, `list` will point to the next node (which has a larger `tosort` element).

`void` [Function]
`gal_linkedlist_osll_into_sll` (*struct gal_linkedlist_osll* `*in`, *struct gal_linkedlist_sll* `**out`)

Convert the orderd `size_t` linked list into an ordinary `size_t` linked list (no longer sorted). This can be useful when all the elements have been added and you just need to pop-out elements.

`gal_linkedlist_tosll` [Structure]

Two-way, ordered `size_t` linked list (`tosll`. Each node in this Doubly-linked list contains a `size_t` value and a floating point value. The floating point value is used as a reference to add new nodes in a sorted manner. In the functions here, this linked list can be pointed to by two pointers (largest and smallest) with the following format:

```
                largest pointer
                |
       NULL <-- (v0,s0) <--> (v1,s1) <--> ... (vn,sn) --> NULL
                                                 |
                                      smallest pointer
```

At any moment, the two pointers will point the nodes containing the "largest" and "smallest" values and the rest of the nodes will be sorted. This is useful when an unknown number of nodes are being added continuously and during the operations it is important to have the nodes in a sorted format.

```
struct gal_linkedlist_tosll
{
  size_t                        v;
  float                         s;
  struct gal_linkedlist_tosll *prev;
  struct gal_linkedlist_tosll *next;
};
```

`void` [Function]
`gal_linkedlist_print_tosll` (*struct gal_linkedlist_tosll* `*largest`, *struct gal_linkedlist_tosll* `*smallest`)

Print all the values within the linked list to the output terminal. This can be used for inspecting the status of the list during development.

`void` [Function]
`gal_linkedlist_add_to_tosll_end` (*struct gal_linkedlist_tosll* `**largest`, *struct gal_linkedlist_tosll* `**smallest`, *size_t* `value`, *float* `tosort`)

Allocate space for a new node in `list`, and store `value` into it. The new node will be added to the list based on the `s` value as described in the description of `gal_`

`linkedlist_tosll`. If this is the first node to be added to the list, both the `largest` and `smallest` pointers can be `NULL`.

`void` [Function]
`gal_linkedlist_pop_from_tosll_start` (*struct gal_linkedlist_tosll* `**lartest`, *struct gal_linkedlist_tosll* `**smallest`, *size_t* `*value`, *float* `*tosort`)
Pop the node with the smallest `s` of `list`, then put its value in `value` and its sort value in `sortvalue`. This function will also free the allocated space for the popped node and after this function, `largest` and `smallest` will be modified respectively. Note that even though only the smallest pointer will be popped, when there was only one node in the list, the `largest` pointer also has to change, so we need both.

`void` [Function]
`gal_linkedlist_smallest_tosll` (*struct gal_linkedlist_tosll* `*largest`, *struct gal_linkedlist_tosll* `**smallest`)
Given the `largest` pointer, set `smallest` as described in `gal_linkedlist_tosll`.

`void` [Function]
`gal_linkedlist_tosll_into_sll` (*struct gal_linkedlist_tosll* `*in`, *struct gal_linkedlist_sll* `**out`)
Convert the two-way orderd `size_t` linked list into an ordinary `size_t` linked list (no longer sorted). This can be useful when all the elements have been added and you just need to pop-out elements.

`void` [Function]
`gal_linkedlist_tosll_free` (*struct gal_linkedlist_tosll* `*largest`)
Free the space for all the nodes in `largest`. Note that any values within the nodes will also be discarded.

10.2.6 Mesh grid for an image (`mesh.h`)

The basic concepts behind the mesh and channel grids in Gnuastro were explained in Section 6.5.2 [Tiling an image], page 119. Besides the improved measurement purposes introduced there, tiling an image can also greatly speed up some processing operations over the image, since each mesh can be given to a different thread and so the CPU can be working on multiple regions of the image simultaneously. The mesh structure and functions introduced here are declared in `gnuastro/mesh.h` and allow you to also use operate on an image mesh.

Please note that as discussed in Section 10.2 [Gnuastro library], page 197, most library functions will significantly evolve with future releases. The mesh structure and functions introduced here are high on this list: until now they were only internally used and so not much attention had been paid to their usability outside Gnuastro. Gnuastro's `lib/mesh.c` contains the full function definitions and is heavily commented, please also consult that file if the explanations here are not clear enough. However, the basic idea behind this structure, and the functions to manipulate data over it, is a powerful tool that can be generically used in many contexts. So please use the structure and functions here very critically and share your criticisms, thoughts and ideas on them. Hopefully in future releases the structure and functions were will evolve to become an easy to use and generically applicable mesh grid system.

`gal_mesh_params` [Structure]

This structure keeps all the necessary paramters for a particular mesh and channel grid over an image. This structure only keeps the information about the mesh and channel grid structure. It doesn't keep a copy of the different parts of the image. So when working on several images of the same size, you can use the same mesh structure. Its individual elements are too many to fully list here, if you are interested please look into the installed `gnuastro/mesh.h` header.

One important component of the structure which is commonly used in the functions is the `garray`, where each mesh has a unique `gid`. `garray` stands for grid-array. It is used for operations on the mesh grid, where one value is to be assigned for each mesh. It has one element for each mesh in the image. However, due to the (possible) existence of channels (see Section 6.5.2 [Tiling an image], page 119), each channel needs its own contiguous part (group of meshes) in the full `garray`. Each channel has `gs0*gs1` (dimentions of meshes in each channel) elements. There are `nch` parts (or channels) in total. In short, the meshs in each channel have to be contiguous to facilitate the neighbor analysis in interpolation and other channel specific jobs. So, the over-all structure can be viewed as a 3D array. The operations on the meshs might need more than one output, for example the mean and the standard deviation. So we have two `garray`s and two `nearest` arrays (for interpolation).

`size_t` [Function]

`gal_mesh_ch_based_id_from_gid` (*struct gal_mesh_params* `*mp`, *size_t* `gid`)

As discussed in the description of `gal_mesh_params`, there are two interal ways to refer to a mesh (with an ID):

- By default, the mesh IDs are set based on their position within the channels (so the meshs in each channel are contiguous). We call this the channel-based mesh ID. The `cgarray1` and `cgarray2` in `gal_mesh_params` store the mesh values based on this ID. All the basic mesh properties, like their types and height and widths are stored based on this ID.
- If the user asks for a full image interpolation or smoothing, then two new arrays are created called `fgarray1` and `fgarray2`. These arrays keep the mesh values as they appear on the image, irrespective of which channel they belong to. We call this the image-based ID. The image based ID is contiguous over the image.

The `garray1` and `garray2` pointers can point to either the `cgarrays` or the `fgarrays`. Ideally, the user should not worry about what garray is pointing to. The caller knows the type of IDs they are using, but they don't know what to put into `garray`. We call the ID that should be put into garray the channel-based ID. It can be either of the two kinds above. So we have the following two functions `gal_mesh_ch_based_id_from_gid` and `gal_mesh_gid_from_ch_based_id` for changing between these IDs.

The former (this function) is used when you are going over the elements in garray (and you are completley ignorant to which one of cgarrays or fgarrays garray points to) and you need the channel based IDs to get basic mesh information like the mesh type and size.

`size_t` [Function]
`gal_mesh_gid_from_ch_based_id` (*struct gal_mesh_params* `*mp`, *size_t* `chbasedid`)

See the explanations in `gal_mesh_ch_based_id_from_gid` first. This function is used when you are going over the meshs through the channel-based IDs, but you need to know what ID to use for `garray`.

`size_t` [Function]
`gal_mesh_img_xy_to_mesh_id` (*struct gal_mesh_params* `*mp`, *size_t* `x`, *size_t* `y`)

You have a pixel's position as `x` and `y` in the final image (in C-based dimentions) and want to know the id to be used in the `garray`s to get a value for this pixel in the mesh-grid. This function will return that ID. As an example, you can get the mesh value at a specific position in the image with:

```
mp->garray[ gal_mesh_img_xy_to_mesh_id(mp, x, y) ];
```

`void` [Function]
`gal_mesh_check_mesh_id` (*struct gal_mesh_params* `*mp`, *long* `**out`)

Save the meshid of each pixel into an array the size of the original image. All the pixels over the mesh will get the value of the mesh ID. The output array `out` can then be written to a FITS file for viewing. This is mainly used for checking/debugging purposes, hence the `check` in the name.

`void` [Function]
`gal_mesh_check_garray` (*struct gal_mesh_params* `*mp`, *float* `**out1`, *float* `**out2`)

Save the values in the `garray`s of the input `mp` structure into two arrays the size of the input image. All the pixels over each mesh will get the value corresponding to that mesh in the `garray`s. The output arrays can then be written to a FITS file for viewing. This is mainly used for checking/debugging purposes, hence the `check` in the name. `out2` will only be set if there are two `garray`s.

`void` [Function]
`gal_mesh_value_file` (*struct gal_mesh_params* `*mp`, *char* `*filename`, *char* `*extname1`, *char* `*extname2`, *struct wcsprm* `*wcs`, *char* `*spack_string`)

Save the mesh grid values into the `filename` FITS file with `extname1` and `extname2` extension names (for the two possible `garrays`), a WCS structure may also optionally be given and `spack_string` is the name of your program to be written in the header. The output FITS file will have one pixel for each mesh, so its dimentions will be significantly smaller than the input image. If you want the mesh values in the same dimention as the input image, use `gal_mesh_check_garray`.

`void` [Function]
`gal_mesh_full_garray` (*struct gal_mesh_params* `*mp`, *int* `reverse`)

As described in `gal_mesh_ch_based_id_from_gid`, the there can be two arrays that keep a value for each mesh in `gal_mesh_params`: `cgarray` and `fgarray`. When `reverse==0`, then this function will fill in the `fgarray`s using the `cgarray`s and vice-versa when `reverse==1`.

`void` [Function]
`gal_mesh_make_mesh` (*struct gal_mesh_params* `*mp`)

Make the mesh structure using the basic information that must already be present within it. The complete list of necessary parameters are as follows, but note that some might not be necessary for your desired funcionality. `s0`, `s1`, `ks0`, `ks1`, `nch1`, `nch2`, `kernel`, `img`, `params`, `minmodeq`, `mirrordist`, `fullsmooth`, `numnearest`, `smoothwidth`, `lastmeshfrac`, `meshbasedcheck`, `interponlyblank`, `fullinterpolation`, `numthreads`. See the installed `gnuastro/mesh.h` for a comment on each parameter within the `gal_mesh_params` structure.

`void` [Function]
`gal_mesh_free_mesh` (*struct gal_mesh_params* `*mp`)

Free all the allocated spaces within the `gal_mesh_params` structure.

`void` [Function]
`gal_mesh_operate_on_mesh` (*struct gal_mesh_params* `*mp`, *void* `*(*meshfunc)(void *)`, *size_t* `oneforallsize`, *int* `makegarray2`, *int* `initialize`)

Run the `meshfunc` function on each mesh in parallel. As the declaration above shows, `meshfunc` should only take one `void *` input and return one `void *` output. This function will setup a pthread environment and make `mp->numthreads` (a number!) calls `meshfunc` which will work on each mesh. `gal_mesh_operate_on_mesh` will return when all the threads have finished their work. If `meshfunc` needs to store more than one value for each mesh, then set `makegarray2=1`. When `initialize==1`, the `garray`(s) will be initialized to NaN.

This function is commonly used within Gnuastro (mainly in NoiseChisel, see Section 7.2 [NoiseChisel], page 136), so search for its applications in the programs for some real-world examples. You can do this with the following command in the top Gnuastro source directory (after unpacking the tarball):

```
grep -r gal_mesh_operate_on_mesh ./bin/*
```

`void` [Function]
`gal_mesh_interpolate` (*struct gal_mesh_params* `*mp`, *char* `*errstart`)

Interpolate the mesh values (`garrays`). After you have run a function on all the meshes with `gal_mesh_operate_on_mesh`, some meshes might not get any values. This function will fill the meshes that didn't get any values (have a value of NaN) using the meshes that received a value. The interpolation is done based on the nearest neighbors, specified within the `gal_mesh_params` structure.

`void` [Function]
`gal_mesh_smooth` (*struct gal_mesh_params* `*mp`)

Smooth the mesh values arrays based on the parameters in `gal_mesh_params`.

`void` [Function]
`gal_mesh_spatial_convolve_on_mesh` (*struct gal_mesh_params* `*mp`, *float* `**conv`)

Do spatial convolution (see Section 6.3.1 [Spatial domain convolution], page 89) on each channel of the mesh grid indendently, therefore two adjacent pixels within the

image that are not in the same channel will not affect eachother during the convolution.

`void` [Function]
`gal_mesh_change_to_full_convolution` (*struct gal_mesh_params* `*mp`, *float* `*conv`)

After running `gal_mesh_spatial_convolve_on_mesh`, it might become necessary to change the convolution into a full image convolution. This function will do the convolution process only on the regions near the channel borders that need to be changed, the rest of the pixels will not be changed by this function.

10.2.7 Polygons (`polygon.h`)

Polygons are commonly necessary in image processing. In Gnuastro, they are used in ImageCrop (see Section 6.1 [ImageCrop], page 75) for cutting out non-rectangular regions of a image. ImageWarp (see Section 6.4 [ImageWarp], page 109) uses them to warp the images into a new pixel grid. The polygon related Gnuastro library macros and functions are introduced here.

In all the functions here the vertices (and points) are defined as an array. So a polygon with 4 vertices will be identified with an array of 8 elements with the first two elements keeping the 2D coordinates of the first vertice and so on.

`GAL_POLYGON_MAX_CORNERS` [Macro]

The largest number of vertices a polygon can have in this library.

`GAL_POLYGON_ROUND_ERR` [Macro]

We have to consider floating point round-off errors when dealing with polygons. For example we will take `A` as the maximum of `A` and `B` when `A>B-GAL_POLYGON_ROUND_ERR`.

`void` [Function]
`gal_polygon_ordered_corners` (*double* `*in`, *size_t* `n`, *size_t* `*ordinds`)

We have a simple polygon (that can result from projection, so its edges don't collide or it doesn't have holes) and we want to order its corners in an anticlockwise fashion. This is necessary for clipping it and finding its area later. The input vertices can have practically any order.

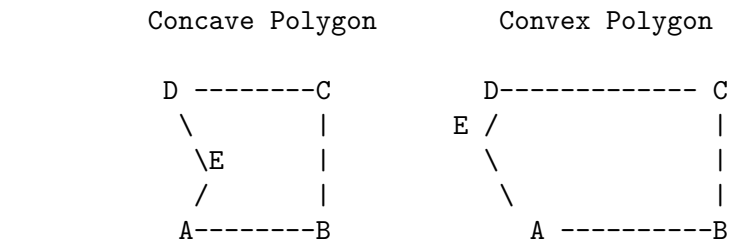
The input (`in`) is an array containing the coordinates (two values) of each vertice. `n` is the number of corners. So `in` should have `2*n` elements. The output (`ordinds`) is an array with `n` elements specifying the indexs in order. This array must have been allocated before calling this function. The indexes are output for more generic usage, for example in a homographic transform (necessary in warping an image, see Section 6.4.1 [Warping basics], page 110), the necessary order of vertices is the same for all the pixels. In other words, only the positions of the vertices change, not the way they need to be ordered. Therefore, this function would only be necessary once.

As a summary, the input is unchanged, only `n` values will be put in the `ordinds` array. Such that calling the input coordinates in the following fashion will give an anti-clockwise order when there are 4 vertices:

```
1st vertice: in[ordinds[0]*2], in[ordinds[0]*2+1]
```

```
2nd vertice: in[ordinds[1]*2], in[ordinds[1]*2+1]
3rd vertice: in[ordinds[2]*2], in[ordinds[2]*2+1]
4th vertice: in[ordinds[3]*2], in[ordinds[3]*2+1]
```

The implementation of this is very similar to the Graham scan in finding the Convex Hull. However, in projection we will never have a concave polygon (the left condition below, where this algorithm will get to E before D), we will always have a convex polygon (right case) or E won't exist!



This is because we are always going to be calculating the area of the overlap between a quadrilateral and the pixel grid or the quadrilaterral its self.

The `GAL_POLYGON_MAX_CORNERS` macro is defined so there will be no need to allocate these temporary arrays seprately. Since we are dealing with pixels, the polygon can't really have too many vertices.

`double` [Function]
`gal_polygon_area` (*double* `*v`, *size_t* `n`)

Find the area of a polygon with vertices defined in `v`. `v` points to an array of doubles which keep the positions of the vertices such that `v[0]` and `v[1]` are the positions of the first vertice to be considered.

`int` [Function]
`gal_polygon_pin` (*double* `*v`, *double* `*p`, *size_t* `n`)

Return `1` if the point `p` is within the polygon whose vertices are defined by `v` and `0` otherwise. Note that the vertices of the polygon have to be sorted in an anti-clock-wise manner.

`int` [Function]
`gal_polygon_ppropin` (*double* `*v`, *double* `*p`, *size_t* `n`)

Similar to `gal_polygon_pin`, except that if the point `p` is on one of the edges of a polygon, this will return `0`.

`void` [Function]
`gal_polygon_clip` (*double* `*s`, *size_t* `n`, *double* `*c`, *size_t* `m`, *double* `*o`, *size_t* `*numcrn`)

Clip (find the overlap of) two polygons. This function uses the Sutherland-Hodgman (`https://en.wikipedia.org/wiki/Sutherland%E2%80%93Hodgman_algorithm`) polygon clipping algorithm. Note that the vertices of both polygons have to be sorted in an anti-clock-wise manner.

10.2.8 Qsort functions (`qsort.h`)

The C programming language comes with the `qsort` (Quick sort) function. By allowing you to define the sorting based on the data, `qsort` is a generic function which allows you to

sort any kind of data structure. The functions here are all meant to be passed onto `qsort` for sorting the respective types of data in the respective manner. Because of this all these functions have the same two argument types.

`gal_qsort_index_arr` [Global variable]
Pointer to a floating point array (`float *`) to use as a reference in `gal_qsort_index_float_decreasing`, see the explanation there for more.

`int` [Function]
`gal_qsort_int_decreasing` (*const void* `*a`, *const void* `*b`)
When passed to `qsort`, this function will sort an integer array in decreasing order.

`int` [Function]
`gal_qsort_int_increasing` (*const void* `*a`, *const void* `*b`)
When passed to `qsort`, this function will sort an integer array in increasing order.

`int` [Function]
`gal_qsort_float_decreasing` (*const void* `*a`, *const void* `*b`)
When passed to `qsort`, this function will sort a single precision floating point array in decreasing order.

`int` [Function]
`gal_qsort_float_increasing` (*const void* `*a`, *const void* `*b`)
When passed to `qsort`, this function will sort a single precision floating point array in increasing order.

`int` [Function]
`gal_qsort_double_decreasing` (*const void* `*a`, *const void* `*b`)
When passed to `qsort`, this function will sort a double precision floating point array in decreasing order.

`int` [Function]
`gal_qsort_double_increasing` (*const void* `*a`, *const void* `*b`)
When passed to `qsort`, this function will sort a double precision floating point array in increasing order.

`int` [Function]
`gal_qsort_index_float_decreasing` (*const void* `*a`, *const void* `*b`)
When passed to `qsort`, this function will sort a `size_t` array based on decreasing values in the `gal_qsort_index_arr` single precision floating poiny array. The floating point array will not be changed, it is only read. For example, if we have the following source code:

```
#include <stdio.h>
#include <stdlib.h>           /* qsort is defined in stdlib.h. */
#include <gnuastro/qsort.h>

int
main (void)
{
```

```
  size_t s[4]={0, 1, 2, 3};
  float f[4]={1.3,0.2,1.8,0.1};
  gal_qsort_index_arr=f;
  qsort(s, 4, sizeof(size_t), gal_qsort_index_float_decreasing);
  printf("%lu, %lu, %lu, %lu\n", s[0], s[1], s[2], s[3]);
  return EXIT_SUCCESS;
}
```

The output will be: `2, 0, 1, 3`.

10.2.9 Spatial convolution (`spatialconvolve.h`)

Convolution is a common operation in data processing and is fully explained in Section 6.3.1 [Spatial domain convolution], page 89, as part of Section 6.3 [Convolve], page 88. We will therefore not explain the details, or its purpose here. The main high-level function here is `gal_spatialconvolve_convolve` which will do spatial convolution on multiple threads. The structure and functions introduced here are declared in `gnuastro/spatialconvolve.h`.

In the first release of the Gnuastro libraries, this is the most high-level library since it was needed by multiple programs. In future releases this library will most probably merge with the frequency domain convolution functions to allow programmers to choose from the type of convolution they want. The main

`gal_spatialconvolve_params` [Structure]
: This structure keeps all the basic convolution information so subsequent calls can be simpler. Please see the header for the details of its elements, complemented with comments.

`void` [Function]
`gal_spatialconvolve_pparams` (*float* `*input`, *size_t* `is0`, *size_t* `is1`, *float* `*kernel`, *size_t* `ks0`, *size_t* `ks1`, *size_t* `nt`, *int* `edgecorrection`, *float* `*out`, *size_t* `*indexs`, *struct gal_spatialconvolve_params* `*scp`)
: Initialize `gal_spatialconvolve_params` with the given arguments.

`void *` [Function]
`gal_spatialconvolve_thread` (*void* `*inparam`)
: This function should be passed onto `pthread_create` to create a new thread for convolutionn on a region of the image.

`void` [Function]
`gal_spatialconvolve_convolve` (*float* `*input`, *size_t* `is0`, *size_t* `is1`, *float* `*kernel`, *size_t* `ks0`, *size_t* `ks1`, *size_t* `numthreads`, *int* `edgecorrection`, *float* `**out`)
: Convolve `input` (with `is0*is1` pixels) with the `kernel` (with `ks0*ks1`) on `numthreads` CPU threads. If you want edge-correction (see Section 6.3.1.2 [Edges in the spatial domain], page 90), then set `edgecorrection` to `1`, otherwise set it to `0`. Memory will be allocated for the output and the convolved image will be stored in `out`.

10.2.10 Statistical operations (`statistics.h`)

This header includes a large variety of very basic statistical operators for various data types. Please note that we expect to greatly simplify the functions here for easier and more general usage in future releases. Ideally, they should be completely removed and the GNU Scientific Library's functions should be used. Since these functions are used in various parts of Gnuastro for multiple purposes, in this first library release (Gnuastro 0.2), there might be parallels, or non-homogenous arguments. Such situations arise here because of the history of Gnuastro: the libraries gew out of the programs, so it will take a little while to correct.

`GAL_STATISTICS_MAX_SIG_CLIP_CONVERGE` [Structure]
The maximum number of times to try for σ-clipping (see Section 7.1.2 [Sigma clipping], page 129).

`void` [Function]
`gal_statistics_long_non_blank_min` (*long* `*in`, *size_t* `size`, *long* `*min`, *long* `blank`)
Find the the minimum (non-blank) value in the `in` array (with `size` elements). The long type doesn't have a NaN value like the float types, see Section 6.1.3 [Blank pixels], page 77. So as blank pixels, a value in the range of acceptable values (`blank` must be given so it is explictly ignored. You can use `GAL_FITS_LONG_BLANK` in `gnuastro/fits.h`.

`void` [Function]
`gal_statistics_long_non_blank_max` (*long* `*in`, *size_t* `size`, *long* `*max`, *long* `blank`)
Similar to `gal_statistics_long_non_blank_min`, but find the maximum.

`void` [Function]
`gal_statistics_float_min` (*float* `*in`, *size_t* `size`, *float* `*min`)
Find the minimum value in the single precision floating point `in` array (with `size` elements) and store the value in `min`. Note that for all floating point types, any blank (NaN) value will be automatically ignored in this function.

`void` [Function]
`gal_statistics_float_max` (*float* `*in`, *size_t* `size`, *float* `*max`)
Similar to `gal_statistics_float_min` but find the maximum.

`void` [Function]
`gal_statistics_double_min` (*double* `*in`, *size_t* `size`, *double* `*min`)
Similar to `gal_statistics_float_min` but for double precision floating point types.

`double` [Function]
`gal_statistics_double_min_return` (*double* `*in`, *size_t* `size`)
Similar to `gal_statistics_double_min` but minmium will be returned, not put in a pointer.

`void` [Function]
`gal_statistics_double_max` (*double* `*in`, *size_t* `size`, *double* `*max`)
Similar to `gal_statistics_double_min`, but find maximum value.

`double` [Function]
`gal_statistics_double_max_return` (*double* `*in`, *size_t* `size`)
Similar to `gal_statistics_double_max_return`, but return maximum.

`void` [Function]
`gal_statistics_float_max_masked` (*float* `*in`, *unsigned char* `*mask`, *size_t* `size`, *float* `*max`)
Only find maximum where `mask` has a value of `0`.

`void` [Function]
`gal_statistics_float_second_max` (*float* `*in`, *size_t* `size`, *float* `*secondmax`)
Find the second largest value in the array.

`void` [Function]
`gal_statistics_float_second_min` (*float* `*in`, *size_t* `size`, *float* `*secondmin`)
Find the second smallest value in the array.

`void` [Function]
`gal_statistics_f_min_max` (*float* `*in`, *size_t* `size`, *float* `*min`, *float* `*max`)
Find the minimum and maximum simultaneously for single precision floating poitn types.

`void` [Function]
`gal_statistics_d_min_max` (*double* `*in`, *size_t* `size`, *double* `*min`, *double* `*max`)
Similar to `gal_statistics_f_min_max`, but for double precision floating point types.

`void` [Function]
`gal_statistics_f_max_with_index` (*float* `*in`, *size_t* `size`, *float* `*max`, *size_t* `*index`)
Find the maximum value in the array along with its index for a single precision floating point array.

`void` [Function]
`gal_statistics_d_max_with_index` (*double* `*in`, *size_t* `size`, *double* `*max`, *size_t* `*index`)
Similar to `gal_statistics_f_max_with_index`, but for a double precision floating point array.

`void` [Function]
`gal_statistics_d_min_with_index` (*double* `*in`, *size_t* `size`, *double* `*min`, *size_t* `*index`)

`void` [Function]
`gal_statistics_f_min_with_index` (*float* `*in`, *size_t* `size`, *float* `*min`, *size_t* `*index`)

`float` [Function]
`gal_statistics_float_sum` (*float* `*in`, *size_t* `size`)
Return the sum of the single precision floating point arrays. Note that NaN values will be ignored.

`float` [Function]
`gal_statistics_float_sum_num` (*float* `*in`, *size_t* `*size`)
Return the sum of elements in the floating point array along with the number of used (non-Nan) elements.

`float` [Function]
`gal_statistics_float_sum_squared` (*float* `*in`, *size_t* `size`)
Return the sum of squares of non-NaN elements in the array.

`float` [Function]
`gal_statistics_float_sum_mask` (*float* `*in`, *unsigned char* `*mask`, *size_t* `size`, *size_t* `*nsize`)
Given a mask (`mask`), return the sum of elements with a `mask` value of `0` (masked pixels are considered to have a non-zero value) and put the number of used elements in `nsize`.

`float` [Function]
`gal_statistics_float_sum_mask_l` (*float* `*in`, *long* `*mask`, *size_t* `size`, *size_t* `*nsize`)
Similar to `gal_statistics_float_sum_mask`, but when the mask is a long type.

`float` [Function]
`gal_statistics_float_sum_squared_mask` (*float* `*in`, *unsigned char* `*mask`, *size_t* `size`, *size_t* `*nsize`)
Return the sum of the squared elements in `in` that are not masked (corresponding element in `mask` is `0`.

`float` [Function]
`gal_statistics_float_sum_squared_mask_l` (*float* `*in`, *long* `*mask`, *size_t* `size`, *size_t* `*nsize`)
Similar to `gal_statistics_float_sum_squared_mask` but when the mask is a long type.

`float` [Function]
`gal_statistics_float_average` (*float* `*in`, *size_t* `size`)
Return the average of the single precision floating point array `in` with `size` elements.

`double` [Function]
`gal_statistics_double_average` (*double* `*in`, *size_t* `size`)
Return the average of the double precision floating point array `in` with `size` elements.

`void` [Function]
`gal_statistics_f_ave` (*float* `*in`, *size_t* `size`, *float* `*ave`, *unsigned char* `*mask`)
Return the average of the single precision floating point array `in` with `size` elements, do not use those elements where `mask` is non-zero.

`void` [Function]
`gal_statistics_f_ave_l` (*float* `*in`, *size_t* `size`, *float* `*ave`, *long* `*mask`)
Similar to `gal_statistics_f_ave`, but when the mask is of type long.

`void` [Function]
`gal_statistics_f_ave_std` (*float* `*in`, *size_t* `size`, *float* `*ave`, *float* `*std`, *unsigned char* `*mask`)

Find the average and standard deviation of the `in` array for pixels where `mask` has value of `0`.

`void` [Function]
`gal_statistics_f_ave_std_l` (*float* `*in`, *size_t* `size`, *float* `*ave`, *float* `*std`, *long* `*mask`)

Similar to `gal_statistics_f_ave_std`, but when the mask has a type of long.

`float` [Function]
`gal_statistics_median` (*float* `*array`, *size_t* `insize`)

Return the median value of a single precision floating point array `array` (which has `insize` elements).

`double` [Function]
`gal_statistics_median_double_in_place` (*double* `*array`, *size_t* `insize`)

Return the median value of a double precision floating point array `array` (which has `insize` elements). After this function, the elements of `array` are also sorted.

`void` [Function]
`gal_statistics_set_bins` (*float* `*sorted`, *size_t* `size`, *size_t* `numbins`, *float* `min`, *float* `max`, *float* `onebinvalue`, *float* `quant`, *float* `**obins`)

Given the input `sorted` array with `size` elements, a given number of `numbins`, an optional `min` and `max`, find the the bin starting points and keep them in `obins`. `obins` has two columns, and this function only fills the first column. The second will be filled by the histogram or cumulative distribution plots.

If `onebinvalue` is NaN, then the bins above will be reported. However, if it is not NaN, all the bins will be shifted such that the lower value of one of the bins will be placed on this value of this variable (if it is in the range of the data). When `min==max`, the actual data range will be used. When `quant` is not zero (let's say it has the value q, for quantile) and `min==max` then the automatic range finder will use the values at the quantiles q and $1-q$.

`void` [Function]
`gal_statistics_histogram` (*float* `*sorted`, *size_t* `size`, *float* `*bins`, *size_t* `numbins`, *int* `normhist`, *int* `maxhistone`)

Build the histogram of the data (`sorted`, with `size` elements) in the `bins` array which also contains the starting bin values and has `numbins` rows. See `gal_statistics_set_bins`.

When `normhist` is non-zero, the histogram will be normalized so the sum of all the bin values is unity. When `maxhistone` is non-zero, the histogram is scaled such that the bin with the largest value gets a value of 1.

`void` [Function]
`gal_statistics_cumulative_fp` (*float* `*sorted`, *size_t* `size`, *float* `*bins`, *size_t* `numbins`, *int* `normcfp`)

Build the cumulative frequency plot of the data (`sorted`, with `size` elements) in the `bins` array which also contains the starting bin values and has `numbins` rows. See `gal_statistics_set_bins`. When `normcfp` is non-zero, the cumulative frequency plot will be normalized so the last bin has a value of 1.

`void` [Function]
`gal_statistics_save_hist` (*float* `*sorted`, *size_t* `size`, *size_t* `numbins`, *char* `*filename`, *char* `*comment`)

Create a histogram of the data in `sorted` (and `size` elements) with `numbins` and save it into an ASCII text file named `filename` with a possible set of comments (pointed to by `comment` before he table.

`size_t` [Function]
`gal_statistics_index_from_quantile` (*size_t* `size`, *float* `quant`)

Given the number of elements (`size`), return the index corresponding to the quantile `quant`.

`int` [Function]
`gal_statistics_sigma_clip_converge` (*float* `*array`, *int* `o1_n0`, *size_t* `num_elem`, *float* `sigma_multiple`, *float* `accuracy`, *float* `*outave`, *float* `*outmed`, *float* `*outstd`, *int* `print`)

Calculate the σ-clipped (by convergence) average, median and standard deviation on the array `array` with `num_elem` elements (see Section 7.1.2 [Sigma clipping], page 129). If `o1_n0==1`, then it is assumed that the input array is sorted. `sigma_multiple` is the multiple of σ used in the σ-clipping. If the σ-clipping did converge, then this function will return `0`, otherwise, it will return `1`. If `print` is non-zero, then the average, median and standard deviation of each step are printed.

`int` [Function]
`gal_statistics_sigma_clip_certain_num` (*float* `*array`, *int* `o1_n0`, *size_t* `num_elem`, *float* `sigma_multiple`, *size_t* `numtimes`, *float* `*outave`, *float* `*outmed`, *float* `*outstd`, *int* `print`)

Calculate the σ-clipped (with a fixed number of times) average, median and standard deviation on the array `array` with `num_elem` elements (see Section 7.1.2 [Sigma clipping], page 129). If `o1_n0==1`, then it is assumed that the input array is sorted. `sigma_multiple` is the multiple of σ and `numtimes` is the number of times σ-clipping is done. If the σ-clipping was normally done, then this function will return `0`, otherwise, it will return `1`. If `print` is non-zero, then the average, median and standard deviation of each step are printed.

`void` [Function]
`gal_statistics_remove_outliers_flat_cdf` (*float* `*sorted`, *size_t* `*outsize`)

Remove the outliers in a distribution based on the slope of the cumulative frequency plot (when it becomes too flat). Before this function, `outssize` will point to the number of elements in `sorted`, after this function, it will be the number of elements that are not outliers.

`GAL_STATISTICS_MODE_LOW_QUANTILE` [Macro]
The lower quantile used in calculating the mode.

`GAL_STATISTICS_MODE_HIGH_QUANTILE` [Macro]
The lower quantile used in calculating the mode.

`GAL_STATISTICS_MODE_SYM_GOOD` [Macro]
The acceptable value for the mode calculation symmetry.

`GAL_STATISTICS_MODE_LOW_QUANT_GOOD` [Macro]
The lowest quantile that an acceptable mode can have.

`GAL_STATISTICS_MODE_SYMMETRICITY_LOW_QUANT` [Macro]
Lower quantile for calculating the symmetricity.

`gal_statistics_mode_params` [Structure]
The input parameters for the mode functions, this is mainly an internal structure and will become hidden in future releases.

`void` [Function]
`gal_statistics_mode_mirror_plots` (*float* `*sorted`, *size_t* `size`, *size_t* `mirrorindex`, *float* `min`, *float* `max`, *size_t* `numbins`, *char* `*histsname`, *char* `*cfpsname`, *float* `mirrorplotdist`)
Create a mirror plot based on the index given by `mirrorindex`. The mode calculation function procedure is defined in Appendix C of Akhlaghi and Ichikawa (2015, ApJS 220, 1. arXiv:1505.01664).

`float` [Function]
`gal_statistics_mode_value_from_sym` (*float* `*sorted`, *size_t* `size`, *size_t* `modeindex`, *float* `sym`)
It happens that you have the symmetricity and you want the flux value at that point, this function will do that job. The mode calculation function procedure is defined in Appendix C of Akhlaghi and Ichikawa (2015, ApJS 220, 1. arXiv:1505.01664).

`void` [Function]
`gal_statistics_mode_index_in_sorted` (*float* `*sorted`, *size_t* `size`, *float* `errorstdm`, *size_t* `*modeindex`, *float* `*modesym`)
Find the quantile of the mode of a sorted distribution. The return value is either `0` (not accurate) or `1` (accurate). The mode calculation function procedure is defined in Appendix C of Akhlaghi and Ichikawa (2015, ApJS 220, 1. arXiv:1505.01664).

10.2.11 Multithreaded programming (`threads.h`)

In modern times, newer CPU generations don't have significantly higher frequencies any more. However, CPUs are being manufactured with more cores, enabling more than one operation (thread) at each instant. This can be very useful to speed up many apsects of processing and in particular image processing.

Most of the programs in Gnuastro utilize multi-threaded programming for the CPU intensive processing steps. This can potentially lead to a significant decrease in the running time of a program, see Section 4.3.1 [A note on threads], page 54. In terms of reading

the code, you don't need to know anything about multi-threaded programming. You can simply follow the case where only one thread is to be used. In these cases, threads are not used and can be completely ignored.

At the time K&R's book was written, using threads was not common, so C's threading capabilities aren't introduced there. We use POSIX threads for multi-threaded programming, defined in the `pthread.h` system wide header. There are various resources for learning to use POSIX threads, the excellent tutorial from LLNL (`https://computing.llnl.gov/tutorials/pthreads/`) is a very good start. The book 'Advanced programming in the Unix environment'[14], by Richard Stevens and Stephen Rago, Addison-Wesley, 2013 (Third edition) also has two chapters explaining the POSIX thread constructs which can be very helpful.

An alternative to POSIX threads was OpenMP, but POSIX threads are low level, allowing much more control, while being easier to understand, see Section 11.1 [Why C programming language?], page 236. All the situations where threads are used are completely independent with minimal need of coordination between the threads. Such problems are known as "embarrassingly parallel" problems. They are some of the simplest problems to solve with threads and also the ones that benefit most from threads, see the LLNL introduction[15].

One very useful POSIX thread concept is `pthread_barrier`. Unfortunately, it is only an optional feature in the POSIX standard, so some operating systems don't include it. Therefore in Section 10.2.11.1 [Implementation of `pthread_barrier`], page 231, we introduce our own implementation. You can ignore this section if your system has this construct. Following that, we describe the helper functions in this header that can greatly simplify writing a multi-threaded program, see Section 10.2.11.2 [Gnuastro's thread related functions], page 232, for more.

10.2.11.1 Implementation of `pthread_barrier`

One optional feature of the POSIX Threads standard is the `pthread_barrier` concept. It is a very useful high-level construct that allows for independent threads to "wait" behind a "barrier" for the rest after they finish. Barriers can thus greatly simplify the code in a multi-threaded program, so they are heavily used in Gnuastro. However, since its an optional feature in the POSIX standard, some operating systems don't include it. So to make Gnuastro portable, we have written our own implementation of those `pthread_barrier` functions.

At `./configure` time, Gnuastro will check if `pthread_barrier` constructs are available on your system or not. If `pthread_barrier` is not available, our internal implementation will be compiled into the Gnuastro library and the definitions and declarations below will be usable in your code with `#include <gnuastro/threads.h>`.

`pthread_barrierattr_t` [Type]
: Type to specify the attributes of a POSIX threads barrier.

`pthread_barrier_t` [Type]
: Structure defining the POSIX threads barrier.

[14] Don't let the title scare you! The two chapters on Multi-threaded programming are very self-sufficient and don't need any more knowledge than K&R.

[15] `https://computing.llnl.gov/tutorials/parallel_comp/`

`int` [Function]
`pthread_barrier_init` (*pthread_barrier_t* `*b`, *pthread_barrierattr_t* `*attr`, *unsigned int* `limit`)

Initialize the barrier `b`, with the attributes `attr` and total `limit` (a number of) threads that must wait behind it. This function must be called before spinning off threads.

`int` [Function]
`pthread_barrier_wait` (*pthread_barrier_t* `*b`)

This function is called within each thread, just before it is ready to return. Once a thread's function hits this, it will "wait" until all the other functions are also finished.

`int` [Function]
`pthread_barrier_destroy` (*pthread_barrier_t* `*b`)

Destroy all the information in the barrier structure. This should be called by the function that spinned off the threads after all the threads have finished.

> **Destroy a barrier before re-using it:** It is very important to destroy the barrier before (possibly) reusing it. This destroy function not only destroys the internal structures, it also waits (in 1 microsecond intervals, so you will not notice!) until all the threads don't need the barrier structure any more. If you immediately start spinning off new threads with a not-destroyed barrier, then the internal structure of the remaining threads will get mixed with the new ones and you will get very strange and apparently random errors that are extremely hard to debug.

10.2.11.2 Gnuastro's thread related functions

These are some high-level functions in `gnuastro/threads.h` to facilitate programming in with POSIX threads. We have created a simple C program for testing these functions in `tests/lib/multithread.c`. This small program was compiled and run on your system when you ran `make check`. You can use it as a template to easily create small multithreaded programs and efficiently use your powerful CPU.

`GAL_THREADS_NON_THRD_INDEX` [Macro]

This value will be used in the output of `gal_threads_dist_in_threads` as a non-index. It plays the same role as a string's null character: '`\0`': as soon as the parser sees this value, it will stop continuing.

`void` [Function]
`gal_threads_attr_barrier_init` (*pthread_attr_t* `*attr`, *pthread_barrier_t* `*b`, *size_t* `limit`)

Initialize the general thread attribute `attr` and the barrier `b` with `limit` threads to wait behind the barrier. For maximum efficiency, the threads initialized with this function will be detached. Therefore no communication is possible between these threads and in particular `pthread_join` won't work on these threads. You have to use the barrier constructs to wait for all threads to finish.

`void` [Function]
`gal_threads_dist_in_threads` (*size_t* `numactions`, *size_t* `numthreads`, *size_t* `**outthrds`, *size_t* `*outthrdcols`)

Identify the "index"es (starting from 0) of the actions to be done on each thread in the `outthrds` array. `outthrds` is treated as a 2D array with `numthreads` rows and `outthrdcols` columns. The indexs in each row, identify the actions that should be done by one thread. Please see the explanation below to understand the purpose of this operation.

Let's assume you have A actions (where there is only one function and the input values differ for each action) and T threads available to the system with $A > T$ (common values for these two would be $A > 1000$ and $T < 10$). Spinning off a thread is not a cheap job and requires a significant number of CPU cycles. Therefore, creating A threads is not the best way to address such a problem. The most efficient way to manage the actions is such that only T threads are created, and each thread works on a list of actions identified for it in series (one after the other). This way your CPU will get all the actions done with minimal overhead.

The purpose of this function is to do do what we explained above: each row in the `outthrds` array contains the indexs of actions which must be done by one thread. `outthrds` contains `outthrdcols` columns. In using `outthrds`, you don't have to know the number of columns. The `GAL_THREADS_NON_THRD_INDEX` macro has a role very similar to a string's `\0`: every row finishes with this macro, so can easily stop parsing the indexes in the row when you confront it. Please see the example program in `tests/lib/multithread.c` for a demonstration.

10.2.12 Text table into a C array (`txtarray.h`)

One of the most common formats to store and transfer normally sized data are ASCII files (which can be seen on a text editor). During your processing experiments (with the data, or what programmers call "developing") you will also often want to see how your data is being processed and one of the easiest ways to check out out your arrays is to write them out as an ASCII file and check them. The functions here are mainly focused on tables that contain numbers.

`GAL_TXTARRAY_LOG` [Macro]

When an element of the input table cannot be read as a number, the column and row number of the element are stored along with the element into one row of a log file. This macro contains the name of the log-file.

`void` [Function]
`gal_txtarray_txt_to_array` (*char* `*filename`, *double* `**array`, *size_t* `*s0`, *size_t* `*s1`)

Store the ASCII text table in `filename` into the C array `array` which has `s0` rows and `s1` columns. Any blank lines or lines starting with `#` (comments) will be ignored. The column delimiters (separating one column from the previous and next) are ' ' (space), '`,`', 'TAB'.

`void` [Function]
`gal_txtarray_printf_format` (*int* `numcols`, *char* `**fmt`, *int* `*int_cols`, *int* `*accu_cols`, *int* `*space`, *int* `*prec`, *char* `forg`)

Return the `printf` formatting string (`fmt`) for `numcols` columns. Three types of output columns are assumed: integers, normal accuracy floating points and extra-precision floating points. Normally most columns will not need too much accuracy when printing (for example only 3 decimals might be enough). However usually a very limited number of columns will need more precision (like RA and Dec). So to store space (each character is a byte) and also make the table more readable, we allow these two types of floating points.

`int_cols` and `accu_cols` are arrays containing the column numbers (starting from zero) that contain integers and extra-accuracy columns. The ending of the column indexes in `int_cols` and `accu_cols` is defined by a negative number. `space` is a three element array which will keep the number of characters that must be used for the integer, normal-accuracy and extra-accuracy columns. `prec` is a two element array which contains the number of decimas to print for the normal and extra accuracy columns. `forg` (read as 'f-or-g') is the `printf` type for the floating point numbers: either `f` (as in `%f`, which will only print decimals) and `g` (as in `%g`, for either printing decimals or exponentials). See `gal_txtarray_array_to_txt` for more.

`void` [Function]
`gal_txtarray_array_to_txt` (*double* `*array`, *size_t* `s0`, *size_t* `s1`, *char* `*comments`, *int* `*int_cols`, *int* `*accu_cols`, *int* `*space`, *int* `*prec`, *char* `forg`, *const char* `*filename`)

Print the C array `array` (with `s0` rows and `s1` columns) as an ASCII text table in `filename`. The string pointed to by `comments` will be printed at the top of the table. See the explanation of `gal_txtarray_printf_format` for the other arguments.

10.2.13 World Coordinate System (`wcs.h`)

The FITS world coordinate system is a mechanism to give physical values to the data points. In the case of images, it can be used to convert pixel images to celestial coordinates like the right ascension and declination. Gnuastro doesn't have any functions to actually do the transformations, the standard tool for doing this in astronomy is WCSLIB (see Section 3.1.1.3 [WCSLIB], page 26). The functions introduced here are just wrappers around the lower-level functions provided by WCSLIB.

`void` [Function]
`gal_wcs_xy_array_to_radec` (*struct wcsprm* `*wcs`, *double* `*xy`, *double* `*radec`, *size_t* `number`, *size_t* `stride`)

Convert the image coordinates `xy` (a two column array with `number` rows) to world coordinates `wcs` (also a two column array, which has to have been allocated prior to this function) using the conversion parameters in `wcs`. `stride` is the number of columns in a larger array, see below.

It might happen that you have one array which contains both the image and world coordinates along with many more columns. `stride` is the width of that larger array. If such an array doesn't exist and `xy` and `radec` are separately allocated arrays with 2×`number` elements, then just set the value of `stride` to `2`.

`void` [Function]
`gal_wcs_radec_array_to_xy` (*struct wcsprm* `*wcs`, *double* `*radec`, *double* `*xy`, *size_t* `number`, *size_t* `stride`)

Similar to `gal_wcs_xy_array_to_radec` but convert the world coordinates in `radec` to image coordinates in `xy`.

`double` [Function]
`gal_wcs_angular_distance` (*double* `r1`, *double* `d1`, *double* `r2`, *double* `d2`)

Return the angular distance between a point located at (`r1`, `d1`) to (`r2`, `d2`). The distance (along a great circle) on a sphere between two points is calculated with the equation below. Since the pixel sides are usually very small, we won't be using the direct formula:

$$\cos(d) = \sin(d_1)\sin(d_2) + \cos(d_1)\cos(d_2)\cos(r_1 - r_2)$$

Here, we will be using the Haversine formula (`https://en.wikipedia.org/wiki/Haversine_formula`) which better considering floating point errors:

$$\frac{\sin^2(d)}{2} = \sin^2\left(\frac{d_1 - d_2}{2}\right) + \cos(d_1)\cos(d_2)\sin^2\left(\frac{r_1 - r_2}{2}\right)$$

`double` [Function]
`gal_wcs_pixel_area_arcsec2` (*struct wcsprm* `*wcs`)

Given the WCS structure `wcs` calculate the pixel area in arcsec-squared.

11 Developing

The basic idea of GNU Astronomy Utilities is for an interested astronomer to be able to easily understand the code of any of the programs or libraries, be able to modify the code if s/he feels there is an improvement and finally, to be able to add new programs or libraries for their own benefit, and the larger community if they are willing to share it. In short, we hope that at least from the software point of view, the "obscurantist faith in the expert's special skill and in his personal knowledge and authority" can be broken, see Section 1.2 [Science and its tools], page 2. The following software architecture can be one of the most basic and easy to understand for any interested inquirer.

In this chapter, first some general design choices are tackled in `Why C` and `Program design philosophy`. The project management webpage and mailing lists are explained in Section 11.3 [Gnuastro project webpage], page 238, and `Developing mailing lists`. In Section 11.5 [Coding conventions], page 240, Gnuastro's adopted coding conventions are discussed and is followed by Section 11.6 [Program source], page 244, which describes how to easily navigate the source files in each program and also contains a template program to easily add new programs. Some other general aspects are then discussed: Section 11.7 [Documentation], page 247, Section 11.8 [Building and debugging], page 248, Section 11.9 [Test scripts], page 249, and Section 11.10 [Developer's checklist], page 250. This chapter finishes with a full description of Gnuastro's work-flow in Section 11.11 [Contributing to Gnuastro], page 250.

11.1 Why C programming language?

Currently the programming language that is most commonly used in scientific applications is C++, and more recently Python. One of the main reasons behind this choice is that through the Object oriented programming paradigm, they offer a much higher level of abstraction. However, GNU Astronomy Utilities are written in the C programming language. The reasons can be summarized with simplicity, portability and speed. All three are very important in a scientific software.

Simplicity can best be demonstrated in a comparison of the main books of C++ and C. The "C programming language"[1] book, written by the authors of C, is only 286 pages and covers a very good fraction of the language, it has also remained unchanged from 1988. C is the main programming language of nearly all operating systems and there is no plan of any significant update. On the other hand, the most recent "C++ programming language"[2] book, also written by its author, has 1366 pages and its fourth edition came out in 2013! As discussed in Section 1.2 [Science and its tools], page 2, it is very important for other scientists to be able to readily read the code of a program at their will with minimum requirements.

In C++, inheriting objects in the object oriented programming paradigm and their internal functions make the code very easy to write for the programmer who is deeply invested in those objects and understands all their relations well. But it simultaneously makes reading the program for a first time reader (a curious scientist who wants to know only how

[1] Brian Kernighan, Dennis Ritchie. *The C programming language*. Prentice Hall, Inc., Second edition, 1988. It is also commonly known as K&R and is based on the ANSI C and ISO C90 standards.

[2] Bjarne Stroustrup. *The C++ programming language*. Addison-Wesley Professional; 4 edition, 2013.

a small step was done) extremely hard. Before understanding the methods, the scientist has to invest a lot of time in understanding those objects and their relations. But in C all variables can be given as the basic language types for example `int`s or `float`s and their pointers to define arrays. So when an outside reader is only interested in one part of the program, that part is all they have to understand.

Recently it is also becoming common to write scientific software in Python, or a combination of it with C or C++. Python is a high level scripting language which doesn't need compilation. It is very useful when you want to do something on the go and don't want to be halted by the troubles of compiling, linking, memory checking, etc. When the data sets are small and the job is temporary, this ability of Python is great and is highly encouraged. A very good example might be plotting, in which Python is undoubtedly one of the best.

But as the data sets increase in size and the processing becomes very complicated, the speed of Python scripts significantly decrease. So when the program doesn't change too often and is widely used in a large community mostly on large data sets (like astronomical images), using Python will waste a lot of valuable research-hours. It is possible to wrap C or C++ functions with Python to fix the speed issue.

Like C++, Python is object oriented, so as explained above, it needs a high level of experience with that particular program to fully understand its inner workings. To make things worse, since it is mainly for fast and on the go programming, it can undergo significant changes. One recent example is how Python 2.x and Python 3.x are not compatible. Lots of research teams that invested heavily in Python 2.x cannot benefit from Python 3.x or future versions any more. Some converters are available, but since they are automatic, lots of complications might arise in the conversion. Thus, re-writing all the changes would be the only truly reliable option. If a research project begins using Python 3.x today, there is no telling how compatible their investments will be when Python 4.x or 5.x will come out. This stems from the core principles of Python, which are very useful when you look in the 'on the go' basis as described before and not future usage. Future-proof code (as long as current operating systems will be used) will be written in C.

The portability of C is best demonstrated by the fact that both C++ and Python are part of the C-family of programming languages which also include Java, Perl, and many other languages. C libraries can be immediately included in C++ and with it is easily possible to write wrappers for C libraries in programs in all C-family programming languages. This will allow other scientists to benefit from C libraries with any of those languages. With version 0.2, Gnuastro's C library is installed along with the programs and task 13786[3] has been defined to add wrappers for higher level languages.

The final reason was speed. This is another very important aspect of C which is not independent of simplicity (first reason discussed above). The abstractions provided by the higher-level languages (which also makes learning them harder for a newcomer) comes at the cost of speed. Since C is a low-level language[4] (closer to the hardware), it is much less complex for both the human reader and the computer. The former was discussed above in simplicity and the latter helps in making the program run more efficiently (faster). This thus allows for a closer relation between the scientist/programmer (program) and the actual

[3] `http://savannah.gnu.org/task/?13786`

[4] Low-level languages are those that directly operate the hardware like assembly languages. So C is actually a high-level language, but it can be considered one of the lowest-level, high-level languages.

data/processing. The GNU coding standards[5] also encourage the use of C over all other languages when generality of usage and "high speed" is desired.

11.2 Program design philosophy

The core processing functions of each program are written mostly with the basic ISO C90 standard. We do make lots of use of the GNU additions to the C language in the GNU C library, but these additional functions are mainly used in the user interface functions (reading your inputs and preparing them prior to or after the analysis). The actual algorithms, which most scientists would be more interested in, are much more closer to ISO C90. For this reason, the source files containing user interface code and those containing actual processing code are clearly separated, see Section 11.6 [Program source], page 244. If anything particular to the GNU C library is used in the processing functions, it is explained in the comments in between the code.

Similar to GNU Coreutils, all the Gnuastro programs provide very low level operations. This enables you to use the GNU Bash scripting language (which is the default in most GNU/Linux operating systems) or any other shell you might be using to operate on a large number of files or do very complex things through the creative combinations of these tools that the authors had never dreamed of. We have put a few simple examples in Chapter 2 [Tutorials], page 14.

For example all the analysis output is provided as ASCII tables which you can feed into your favorite plotting program to inspect visually. Python's Matplotlib is very useful for fast plotting of the tables to immediately check your results. If you want to include the plots in a document, you can use the PGFplots package within LaTeX, no attempt is made to include such operations in Gnuastro. In short, Bash can act as a glue to connect the inputs and outputs of all these various Gnuastro programs (and other programs) in any fashion you please.

The advantage of this architecture is that the programs become small and transparent: the starting and finishing point of every program is clearly demarcated. For nearly all operations on a modern computer, the read/write speed is very insignificant compared to the actual processing a program does. Therefore the complexity which arises from sharing memory in a large application is simply not worth the speed gain. This basic design is influenced by Eric Raymond's "The Art of Unix Programming"[6] which beautifully describes the design philosophy and practice which lead to the success of Unix-based operating systems[7].

11.3 Gnuastro project webpage

Gnuastro's central management hub (`https://savannah.gnu.org/projects/gnuastro/`)[8] is located on GNU Savannah (`https://savannah.gnu.org/`)[9]. It is the central software development management system for all GNU projects. Through this central hub, you can view the list of activities that the developers are engaged in, their activity on the version

[5] `http://www.gnu.org/prep/standards/`

[6] Eric S. Raymond, 2004, *The Art of Unix Programming*, Addison-Wesley Professional Computing Series.

[7] KISS principle: Keep It Simple, Stupid!

[8] `https://savannah.gnu.org/projects/gnuastro/`

[9] `https://savannah.gnu.org/`

controlled source, and other things. Each defined activity in the development cycle is known as an 'issue' (or 'item'). An issue can be a bug (see Section 1.7 [Report a bug], page 9), or a suggested feature (see Section 1.8 [Suggest new feature], page 11) or an enhancement or generally any *one* job that is to be done. In Savannah, issues are classified into three categories or 'tracker's:

Support
: This tracker is a way that (possibly anonymous) users can get in touch with the Gnuastro developers. It is a complement to the bug-gnuastro mailing list (see Section 1.7 [Report a bug], page 9). Anyone can post an issue to this tracker. The developers will not submit an issue to this list. They will only reassign the issues in this list to the other two trackers if they are valid[10]. Ideally (when the developers have time to put on Gnuastro, please don't forget that Gnuastro is a volunteer effort), there should be no open items in this tracker.

Bugs
: This tracker contains all the known bugs in Gnuastro (problems with the existing tools).

Tasks
: The items in this tracker contain the future plans (or new features/capabilities) that are to be added to Gnuastro.

All the trackers can be browsed by a (possibly anonymous) visitor, but to edit and comment on the Bugs and Tasks trackers, you have to be a registered Gnuastro developer. When posting an issue to a tracker, it is very important to choose the 'Category' and 'Item Group' options accurately. The first contains a list of all Gnuastro's programs along with 'Installation', 'New program' and 'Webpage'. The "Item Group" contains the nature of the issue, for example if it is a 'Crash' in the software (a bug), or a problem in the documentation (also a bug) or a feature request or an enhancement.

The set of horizontal links on the top of the page (Starting with 'Main' and 'Homepage' and finishing with 'News') are the easiest way to access these trackers (and other major aspects of the project) from any part of the project webpage. Hovering your mouse over them will open a drop down menu that will link you to the different things you can do on each tracker (for example, 'Submit new' or 'Browse'). When you browse each tracker, you can use the "Display Criteria" link above the list to limit the displayed issues to what you are interested in. The 'Category' and 'Group Item' (explained above) are a good starting point.

Any new issue that is submitted to any of the trackers, or any comments that are posted for an issue, is directly forwarded to the gnuastro-devel mailing list (`https://lists.gnu.org/mailman/listinfo/gnuastro-devel`, see Section 11.4 [Developing mailing lists], page 240, for more). This will allow anyone interested to be up to date on the overall development activity in Gnuastro and will also provide an alternative (to Savannah) archiving for the development discussions. Therefore, it is not recommended to directly post an email to this mailing list, but do all the activities (for example add new issues, or comment on existing ones) on Savannah.

[10] Some of the issues registered here might be due to a mistake on the user's side, not an actual bug in the program.

Do I need to be a member in Savannah to contribute to Gnuastro? No.

The full version controlled history of Gnuastro is available for anonymous download or cloning. See Section 11.11.3 [Production workflow], page 253, for a description of Gnuastro's Integration-Manager Workflow. In short, you can either send in patches, or make your own fork. If you choose the latter, you can push your changes to your own fork and inform us. We will then pull your changes and merge them into the main project. Please see Section 11.11.4 [Forking tutorial], page 254, for a tutorial.

11.4 Developing mailing lists

To keep the developers and interested users up to date with the activity and discussions within Gnuastro, there are two mailing lists which you can subscribe to:

`gnuastro-devel@gnu.org`
(at `https://lists.gnu.org/mailman/listinfo/gnuastro-devel`)
: All the posts made in the support, bugs and tasks discussions of Section 11.3 [Gnuastro project webpage], page 238, are also sent to this mailing address and archived. By subscribing to this list you can stay up to date with the discussions that are going on between the developers before, during and (possibly) after working on an issue. All discussions are either in the context of bugs or tasks which are done on Savannah and circulated to all interested people through this mailing list. Therefore it is not recommended to post anything directly to this mailing list. Any mail that is sent to it from Savannah to this list has a link under the title "Reply to this item at:". That link will take you directly to the issue discussion page, where you can read the discussion history or join it.

  While you are posting comments on the Savannah issues, be sure to update the meta-data. For example if the task/bug is not assigned to anyone and you would like to take it, change the "Assigned to" box, or if you want to report that it has been applied, change the status and so on. All these changes will also be circulated with the email very clearly.

`gnuastro-commits@gnu.org`
(at `https://lists.gnu.org/mailman/listinfo/gnuastro-commits`)
: This mailing list is defined to circulate all commits that are done in Gnuastro's version controlled source, see Section 3.2.2 [Version controlled source], page 31. If you have any ideas, or suggestions on the commits, please use the bug and task trackers on Savannah to followup the discussion, do not post to this list. All the commits that are made for an already defined issue or task will state the respective ID so you can find it easily.

11.5 Coding conventions

Generally we try our best to follow the GNU coding standards, besides those the following conventions are adhered to until now. If new code is also added in the same manner, it would be much more easily readable by any interested astronomer (who will become familiar with it after reading once).

- It is very important that the code be easy to read by the eye. So when the order of several lines within a function does not matter (mostly when defining variables at the

start of a function). You should put the lines in the order of increasing length and group the variables with similar types such that this half-pyramid of declarations becomes most visible. If the reader is interested, a simple search will show them the variable they are interested in. However, when looking through the functions or reading the separate steps of the functions, this 'order' in the declarations will make reading the rest of the function steps much more easier and pleasant to the eye.

- When ever you see that the function cannot be fully displayed (vertically) in your monitor, this is a sign that it has become too long and should be broken up into multiple functions. 40 lines is usually a good reference. When the start and end of a function are clearly visible in one glance, the function is much more easier to understand. This is most important for low-level functions (which usually define a lot of variables). Low-level functions do most of the processing, they will also be the most interesting part of a program for an inquiring astronomer. This convention is less important for higher level functions that don't define too many variables and whose only purpose is to run the lower-level functions in a specific order and with checks.

  In general you can be very liberal in breaking up the functions into smaller parts, the GNU Compiler Collection (GCC) will automatically compile the functions as inline functions when the optimizations are turned on. So you don't have to worry about decreasing the speed. By default Gnuastro will compile with the `-O3` optimization flag.

- All Gnuastro hand-written text files (C source code, Texinfo documentation source, and version control commit messages) should not exceed **75** characters per line. Monitors today are certainly much wider, but with this limit, reading the functions becomes much more easier. Also for the developers, it allows multiple files (or multiple views of one file) to be displayed beside each other on wide monitors.

  Emacs's buffers are excellent for this capability, setting a buffer width of 80 with '`C-u 80 C-x 3`' will allow you to view and work on several files or different parts of one file using the wide monitors common today. Emacs buffers can also be used as a shell prompt and compile the program (with `M-x compile`), and 80 characters is the default width in most terminal emulators. If you use Emacs, Gnuastro sets the 75 character `fill-column` variable automatically for you, see cartouche below.

  For long comments you can use press `Alt-q` in Emacs to separate them into separate lines automatically. For long literal strings, you can use the fact that in C, two strings immediately after each other are concatenated, for example `"The first part, " "and the second part."` Note the space character in the end of the first part. Since they are now separated, you can easily break a long literal string into several lines and adhere to the maximum 75 character line length policy.

- The headers required by each source file (ending with `.c`) should be defined inside of it. All the headers a program needs should not be stacked in another header to include in all source files (for example `main.h`). Although most 'professional' programmers choose the latter type, Gnuastro is primarily written for inquisitive astronomers (who are generally amateur programmers). This is very useful for readability by a first time reader. `main.h` may only include the header file(s) that define types that the main program structure needs, see `main.h` in Section 11.6 [Program source], page 244. Those particular header files that are included in `main.h` can of course be ignored (not included) in separate source files.

- The headers should be classified (by an empty line) into separate groups:
  1. `#include <config.h>`: This must be the first code line (not commented or blank) in each source file. It sets macros that the GNU Portability Library (Gnulib) will use for the possible additions/modifications to C library headers.
  2. The C library (or GNU C library) header files, for example `stdio.h` or `errno.h`.
  3. Installed library header files, including Gnuastro's installed headers (for example `cfitsio.h` or `gsl/gsl_rng.h`, or `gnuastro/fits.h`).
  4. Gnuastro's internal headers (that are not installed), for example `commonparams.h`, or Gnulib's headers (which are also not installed) like `nproc.h`.
  5. For programs, the `main.h` file (which is needed by the next group of headers).
  6. That particular program's header files, for example `mkprof.h`, or `noisechisel.h`.

  As much as order does not matter when you include the header of each group, sort them by length, see above.
- All function names, variables, etc should be in lower case. Macros and preprocessor variables should be in upper case.
- Regarding naming of exported header files, functions, variables, macros, and library functions, we adopted similar conventions to those used by the GNU Scientific Library (GSL)[11]. In particular, in order to avoid clashes with the names of functions and variables coming from other libraries the namespace '`gal_`' is prefixed to them. GAL stands for *G*NU *A*stronomy *L*ibrary. Ideally '`gnuastro_`' should have been used, but that is very long.
- All installed header files should be in the `lib/gnuastro` directory (under the top Gnuastro source directory). After installation, they will be put in the `$prefix/include/gnuastro` directory (see Section 3.3.1.2 [Installation directory], page 36, for `$prefix`). Therefore with this convention Gnuastro's headers can be included in internal (to Gnuastro) and external (a library user) source files with the same line

  ```
  # include <gnuastro/headername.h>
  ```

  Note that unlike the GSL convention, the header file names are not prefixed with a '`gal_`'. If a user manually mixes the Gnuastro (or GSL) headers with other headers, compilation will break because the installed headers depend on each other (the `#include` directive, with the `gnuastro/` directory, above is present in the installed headers too). Therefore the '`gal_`' prefix to the file names is redundant[12].
- All installed functions and variables should also include the base name of the file in which they are defined as prefix, using underscores to separate words[13]. The same applies to exported macros, but in upper case. For example in Gnuastro's top source directory, the prototype of function `gal_box_border_from_center` is

[11] `https://www.gnu.org/software/gsl/design/gsl-design.html#SEC15`

[12] Note that this applies to GSL's header files too, for example see `eigen/gsl_eigen.h` in GSL's source files. GSL uses the `gsl_` filename prefix to separate the headers that will be installed from those that won't. Therefore this filename prefix a technical internal issue there, which we believe will only confuse the user who doesn't need to get involved with such issues.

[13] The convention to use underscores to separate words, called "snake case" (or "snake_case"). This is also recommended by the GNU coding standards.

in `lib/gnuastro/box.h`, and the macro `GAL_POLYGON_MAX_CORNERS` is defined in `lib/gnuastro/polygon.h`. To find the header file in the installed directories, replace `lib/` with `$prefix/include/` (see Section 3.3.1.2 [Installation directory], page 36, for `$prefix`).

This is necessary to give any user (who is not familiar with the library structure) the ability to follow the code of a function that is not in the program or library they are reading. This convention does make the function names longer (a little harder to write), but the extra documentation it provides convention plays an important role in Gnuastro and is worth the cost.

- There should be no trailing white space in a line. To do this automatically every time you save a file in Emacs, add the following line to your `~/.emacs` file.

```
(add-hook 'before-save-hook 'delete-trailing-whitespace)
```

- There should be no tabs in the indentation[14].

- All similar functions are separated by 5 blank lines to be easily seen to be related in a group when parsing the source code by eye. In Emacs you can use `CTRL-u 5 CTRL-o`.

- One group of functions is separated from another with 20 blank lines. In Emacs you can use `CTRL-u 20 CTRL-o`. Each group of functions has short descriptive title of the functions in that group. This title is surrounded by asterisks (`*`) to make it clearly distinguishable. Such contextual grouping and clear title are very important for easily understanding the code.

The last two conventions are not common and might benefit from a short discussion here. With a good experience in advanced text editor operations, to some extent the last two are redundant for a professional developer. However, recall that Gnuastro aspires to be friendly to un-familiar, and un-experienced (in programming) eyes. In other words, as discussed in Section 1.2 [Science and its tools], page 2, we want the code to appear welcoming to someone who is completely new to coding and only has a scientific curiosity.

Newcomers to coding and development, who are curious enough to venture into the code, will not be using (or have any knowledge of) advanced text editors. They will see the raw code in the webpage or on a simple text editor (like Gedit) as plain text. Trying to learn and understand a file with dense functions that are all spaced with one or two blank lines can be very taunting for a newcomer. But when they scroll through the file and see clear titles and meaningful spaces for similar functions (less, meaningful density), we are helping them find and focus on the part they are most interested in sooner and easier.

[14] If you use Emacs, Gnuastro's `.dir-locals.el` file will automatically never use tabs for indentation. To make this a default in all your Emacs sessions, you can add the following line to your `~/.emacs` file: `(setq-default indent-tabs-mode nil)`

GNU Emacs, the recommended text editor: GNU Emacs is an extensible and easily customizable text editor which many programmers rely on for developing due to its countless features. Among them, it allows specification of certain settings that are applied to a single file or to all files in a directory and its sub-directories. In order to harmonize code coming from different contributors, Gnuastro comes with a `.dir-locals.el` file which automatically configures Emacs to satisfy most of the coding conventions above when you are using it within Gnuastro's directories. Thus, Emacs users can readily start hacking into Gnuastro. If you are new to developing, we strongly recommend this editor. Emacs was the first project released by GNU and is still one of its flagship projects. Some resources can be found at:

Official manual
: At `https://www.gnu.org/software/emacs/manual/emacs.html`. This is a great and very complete manual which is being improved for over 30 years and is the best starting point to learn it. It just requires a little patience and practice, but rest assured that you will be rewarded. If you install Emacs, you also have access to this manual on the command-line with the following command (see Section 4.6.4 [Info], page 60).

```
$ info emacs
```

A guided tour of emacs
: At `https://www.gnu.org/software/emacs/tour/`. A short visual tour of Emacs, officially maintained by the Emacs developers.

Unofficial mini-manual
: At `https://tuhdo.github.io/emacs-tutor.html`. A shorter manual which contains nice animated images of using Emacs.

11.6 Program source

Besides the fact that all the programs share some functions that were explained in Chapter 10 [Libraries], page 189, everything else about each program is completely independent. In this section the conventions used in all the program sources are explained in Section 11.6.1 [Mandatory source code files], page 244. To easily understand the explanations in this section you can use Section 11.6.2 [The TEMPLATE program], page 246, which contains the bare minimum code for one program. This template can also be used to easily add new utilities.

11.6.1 Mandatory source code files

Some programs might need lots of source files and if there is no fixed convention, navigating them can become very hard for a new inquirer into the code. The following source files exist in every program's source directory (which is located in `bin/progname`). For small programs, these files are enough. Larger programs will need more files. In general for writing other source files, please choose filenames that are descriptive.

`main.c`
: Each executable has a `main` function, which is located in `main.c`. Therefore this file in any program's source code will be the starting point. No actual processing functions are to be defined in this file, the function(s) in this file are only meant to connect the most high level steps of each program. Generally,

`main` will first call the top user interface function to read user input and make all the preparations. Then it will pass control to the top processing function for that program. The functions to do both these jobs must be defined in other source files.

`main.h` All the major parameters which will be used in the program must be stored in a structure which is defined in `main.h`. The name of this structure is usually `prognameparams`, for example `imgcropparams`. So `#include "main.h"` will be a staple in all the source codes of the program and most of the functions. Keeping all the major parameters of a program in this structure has the major benefit that most functions will only need one argument: a pointer to this structure. This will significantly facilitate the job of the programmer, the inquirer and the computer. All the programs in Gnuastro are designed to be low-level, small and independent parts, so this structure should not get too large.

The main root structure of a program contains at least two other structures: a structure only keeping parameters for user interface functions, which is also defined in `main.h` and the `commonparams` structure which is defined in `lib/gnuastro/commonparams.h`[15]. The main structure can become very large and since the programmer and inquirer often don't need to be confused with these parameters mixed with the actual processing parameters, they are conveniently defined in another structure which is named `uiparams` and is also defined in `main.h`. It could be defined in `ui.h` (see below) so the main functions remain completely ignorant to it, but its parameters might be needed for reporting input conditions, so it is included as part of the main program structure.

This top root structure is conveniently called `p` (short for parameters) by all the programs. The `uiparams` structure is called `up` (for user parameters) and the `commonparams` structure is called `cp`. With this convention any reader can immediately understand where to look for the definition of one parameter.

With this basic root structure, source code of functions can potentially become full of structure de-reference operators (`->`) which can make the code very unreadable. In order to avoid this, whenever a parameter is used more than a couple of times in a function, a parameter of the same type and with the same name (so it can be searched) as the desired parameter should be defined and put the value of the root structure inside of it in definition time. For example

```
char *hdu=p->cp.hdu;
int verb=p->cp.verb;
```

`args.h` The argument parser structures (which are used by GNU C library's Argp) for each program are defined in `args.h`. They are separate global variables and function definitions that will be used by Argp. We recommend going through the appropriate section in GNU C library to understand their exact meaning, although they should be descriptive and understandable enough by looking at a few of the programs.

[15] `lib/gnuastro/commonparams.h` is a program-specific header and not installed as part of the libraries.

`ui.c, ui.h`

The user interface functions are also a unique set of functions in all the programs, so they are particularly named `ui.c` and `ui.h` in all the programs. Everything related to reading the user input arguments and options, checking the configuration files and checking the consistency of the input parameters before the actual program is run should be done in this file. Since most functions are the same, with only the internal checks and structure parameters differing, we recommend going through several of the examples and structuring your `ui.c` in a similar fashion with the rest of the programs.

The most high-level function in `ui.c` is named `setparams` which accepts `int argc, char *argv[]` and a pointer to the root structure for that program, see the explanation for `main.h`. This is the function that `main` calls. The basic idea of the functions in this file is that the processing functions should need a minimum number of such checks. With this convention an inquirer who only wants to understand only one part (mostly the processing part and not user input details) of the code can easily do so. It also makes all the errors related to input appear before the processing begins which is more convenient for the user.

`progname.c`

The main processing functions in each program which keep the function(s) that `main()` will call are in a file named `progname.c`, for example `imgcrop.c` or `noisechisel.c`. The function within these files which `main()` calls is also named after the program, for example

```
void
imgcrop(struct imgcropparams *p)
```

or

```
void
noisechisel(struct noisechiselparams *p)
```

In this manner, if an inquirer is interested the processing steps, they can immediately come and check this file for the first processing step without having to go through `main.c` first. In most situations, any failure in any step of the programs will result in an informative error message and an immediate abort in the program. So there is no need for return values. Under more complicated situations where a return value might be necessary, `void` will be replaced with an `int` in the examples above.

`cite.h` This file keeps the function to be called if the user runs any of the programs with `--cite`, see Section 4.1.4.2 [Operating modes], page 49.

11.6.2 The TEMPLATE program

In the `Version controlled source`, the `bin/` directory contains the source code for each program in a separate subdirectory. The `bin/TEMPLATE` directory contains the bare-minimum files to create a new program. It can be used to understand the conventions described in Section 11.6 [Program source], page 244, or to easily create a new program.

The extra creativity offered by libraries comes at a cost: you have to actually write your `main` function in the programming language of your choice, and compile it before you can

use it. You will also need to find a way to pass input and output information to the function. So when an operation has well-defined inputs and outputs and is commonly needed, it is much more worthwhile to simply do the work in a program (either by modifying an existing one or creating a new one), instead of the using underlying libraries every time. To define your new program based on this template, just take the following steps:

1. Select a name for your new program (for example `myprog`).
2. Copy the `TEMPLATE` directory to a directory with your program's name:

   ```
   $ cp -R bin/TEMPLATE bin/myprog
   ```
3. Open `configure.ac` in the top Gnuastro source. This file manages the operations that are done when a user runs `./configure`. Going down the file, you will notice repetative parts for each program. Copy one of those and correct the names of the copied program to your new program name. We follow alphabetic ordering here, so please place it correctly. There are multiple places where this has to be done, so be patient and go down to the bottom of the file. Ultimately add `bin/myprog/Makefile` to `AC_CONFIG_FILES`, only here the ordering depends on the length of the name.
4. Open `Makefile.am` in the top Gnuastro source. Similar to the previous step, add your new program similar to all the other utilites.
5. Change `TEMPLATE` to `myprog` in the file names and contents of the files in the `bin/myprog/` directory.
6. Run the following command to re-build the configuration and build system.

   ```
   $ autoreconf -f
   ```

   Your new program will be built the next time you run `./configure` and `make`.

11.7 Documentation

Documentation (this book) is an integral part of Gnuastro. Documentation is not considered a separate project. So, no change is considered valid for implementation unless the respective parts of the book have also been updated. The following procedure can be a good suggestion to take when you have a new idea and are about to start implementing it.

The steps below are not a requirement, the important thing is that when you send the program to be included in Gnuastro, the book and the code have to both be fully up-to-date and compatible and the purpose should be very clearly explained. You can follow any path you choose to do this, the following path was what we found to be most successful during the initial design and implementation steps of Gnuastro.

1. Edit the book and fully explain your desired change, such that your idea is completely embedded in the general context of the book with no sense of discontinuity for a first time reader. This will allow you to plan the idea much more accurately and in the general context of Gnuastro or a particular program. Later on, when you are coding, this general context will significantly help you as a road-map.

   A very important part of this process is the program introduction, which explains the purposes of the program. Before actually starting to code, explain your idea's purpose thoroughly in the start of the program section you wish to add or edit. While actually writing its purpose for a new reader, you will probably get some very valuable ideas that you hadn't thought of before, this has occurred several times during the creation of Gnuastro. If an introduction already exists, embed or blend your idea's purpose with

the existing purposes. We emphasize that doing this is equally useful for you (as the programmer) as it is useful for the user (reader). Recall that the purpose of a program is very important, see Section 11.2 [Program design philosophy], page 238.

As you have already noticed for every program, it is very important that the basics of the science and technique be explained in separate subsections prior to the 'Invoking Programname' subsection. If you are writing a new program or your addition involves a new concept, also include such subsections and explain the concepts so a person completely unfamiliar with the concepts can get a general initial understanding. You don't have to go deep into the details, just enough to get an interested person (with absolutely no background) started. If you feel you can't do that, then you have probably not understood the concept yourself! Have in mind that your only limitation in length is the fatigue of the reader after reading a long text, nothing else.

It might also help if you start implementing your idea in the 'Invoking ProgramName' subsection (explaining the options and arguments you have in mind) at this stage too. Actually starting to write it here will really help you later when you are coding.

2. After you have finished adding your initial intended plan to the book, then start coding your change or new program within the Gnuastro source files. While you are coding, you will notice that somethings should be different from what you wrote in the book (your initial plan). So correct them as you are actually coding.
3. In the end, read the section in the book that you edited completely and see if you didn't miss any change in the coding and to see if the context is fairly continuous for a first time reader (who hasn't seen the book or had known of Gnuastro before you made your change).

11.8 Building and debugging

To build the various programs and libraries in Gnuastro, the GNU build system is used which defines the steps in Section 1.1 [Quick start], page 1. It consists of GNU Autoconf and GNU Automake and GNU Libtool which are collectively known as GNU Autotools. They provide a very portable system to check the environment a program is to be installed on prior to compiling and set the compilation conditions based on the particular user. They also make installing everything in their standard places very easy for the programmer. Most of the small caps files that you see in the top source directory of the tarball are created by these three tools (see Section 3.2.2 [Version controlled source], page 31).

To facilitate the building and testing of your work during development, Gnuastro comes with two scripts: `tmpfs-config-make` and `tests/during-dev.sh`. The former is fully described in Section 3.3.1.4 [Configure and build in RAM], page 41. During development, you would commonly run this command only once (at the start of your work). The latter is designed to be run each time you make a change and want to test your work (with some possible input and output). The script itself is heavily commented and thoroughly discribes the best way to use it, so we won't repeat it here. As a summary: you specify the build directory, an output directory (for the built program to be run in and also contains the inputs), the program's short name and the arguments and options that it should be run with. This script will then build Gnuastro, go to the output directory and run the built executable from there. One option for the output directory might be your desktop, so you

can easily see the output files and delete them when you are finished. The main purpose of these scripts is to keep your source directory clean and facilitate your development.

By default all the programs are compiled with optimization flags for increased speed. A side effect of optimization is that valuable debugging information is lost. All the libraries are also linked as shared libraries by default. Shared libraries further complicate the debugging process and significantly slow down the compilation (the `make` command). So during development it is recommended to configure Gnuastro as follows:

```
$ ./configure --disable-shared CFLAGS="-g -O0"
```

These options to configure are already included in `tmpfs-config-make`, you just have to uncomment them.

In order to understand the building process, you can go through the Autoconf, Automake and Libtool manuals, like all GNU manuals they provide both a great tutorial and technical documentation. The "A small Hello World" section in Automake's manual (in chapter 2) can be a good starting guide after you have read the seperate introductions.

11.9 Test scripts

As explained in Section 3.3.2 [Tests], page 42, for every program some simple tests are written to check the various independent features of the program. All the tests are placed in the `tests/` directory. There is one script (`prepconf.sh`) in this folder and several `Makefile`s. The script is the first 'test' that will be run. It will copy all the configuration files from the various directories to a `.gnuastro` directory which it will make so the various tests can set the default values.

For each program, the tests are placed inside directories with the program name. Each test is written as a shell script. The last line of this script is the test which runs the program with certain parameters. The return value of this script determines the fate of the test, see the "Support for test suites" chapter of the Automake manual for a very nice and complete explanation. In every script, two variables are defined at first: `prog` and `execname`. The first specifies the program name and the second the location of the executable.

The most important thing to have in mind about all the test scripts is that they are run from inside the `tests/` directory in the "build tree". Which can be different from the directory they are stored in (known as the "source tree"). The `tmpfs-config-make` script uses this feature (see Section 3.3.1.4 [Configure and build in RAM], page 41). This distinction is made by GNU Autoconf and Automake (which configure, build and install Gnuastro) so that you can install the program even if you don't have write access to the directory keeping the source files. See the "Parallel build trees (a.k.a VPATH builds)" in the Automake manual for a nice explanation.

Because of this, any possible data that was not generated by other tests (and is thus in the build tree), for example the catalogs in ImageCrop tests, has a `$topsrc` prefix instead of `../` for the build three. This `$topsrc` variable points to the source tree where the script can find the source data (it is defined in `tests/Makefile.am`). The executables and other test products were built in the build tree (where they are being run), so they don't need to be prefixed with that variable. This is also true for images or files that were produced by other tests.

11.10 Developer's checklist

This is a checklist of things to do after applying your changes/additions in Gnuastro:

1. If the change is non-trivial, write test(s) in the `tests/progname/` directory to test the change(s)/addition(s) you have made. Then add their file names to `tests/Makefile.am`.
2. Run `$ make check` to make sure everything is working correctly.
3. Make sure the documentation (this book) is completely up to date with your changes, see Section 11.7 [Documentation], page 247.
4. Commit your change to your issue branch (see Section 11.11.3 [Production workflow], page 253, and Section 11.11.4 [Forking tutorial], page 254) Afterwards, run Autoreconf to generate the appropriate version number:

   ```
   $ autoreconf -f
   ```

5. Finally, to make sure everything will be built, installed and checked correctly run

   ```
   $ make distcheck
   ```

   This command will create a distribution file (ending with `.tar.gz`) and try to compile it in the most general cases, then it will run the tests on what it has built in its own mini-environment. If `$ make distcheck` finishes successfully, then you are safe to send your changes to us to implement or for your own purposes. See Section 11.11.3 [Production workflow], page 253, and Section 11.11.4 [Forking tutorial], page 254.

11.11 Contributing to Gnuastro

You have this great idea or have found a good fix to a problem which you would like to implement in Gnuastro. You have also become familiar with the general design of Gnuastro in the previous sections of this chapter (see Chapter 11 [Developing], page 236) and want to start working on and sharing your new addition/change with the whole community as part of the official release. This is great and your contribution is most welcome. This section and the next (see Section 11.10 [Developer's checklist], page 250) are written in the hope of making it as easy as possible for you to share your great idea with the Gnuastro community.

In this section we discuss the final steps you have to take: legal and technical. From the legal perspective, the copyright of any work you do on Gnuastro has to be assigned to the Free Software Foundation (FSF) and the GNU operating system, or you have to sign a disclaimer. We do this to ensure that Gnuastro can remain free in the future, see Section 11.11.1 [Copyright assignment], page 250. From the technical point of view, in this section we also discuss commit guidelines (Section 11.11.2 [Commit guidelines], page 252) and the general version control workflow of Gnuastro in Section 11.11.3 [Production workflow], page 253, along with a tutorial in Section 11.11.4 [Forking tutorial], page 254.

Recall that before starting the work on your idea, be sure to checkout the bugs and tasks trackers in Section 11.3 [Gnuastro project webpage], page 238, and announce your work there so you don't end up spending time on something others have already worked on, and also to attract similarly interested developers to help you.

11.11.1 Copyright assignment

Gnuastro's copyright is owned by the FSF. Professor Eben Moglen, of the Columbia University Law School has given a nice summary of the reasons for this at `https://www.gnu.`

`org/licenses/why-assign`. Below we are copying it verbatim for self consistency (in case you are offline or reading in print).

> Under US copyright law, which is the law under which most free software programs have historically been first published, there are very substantial procedural advantages to registration of copyright. And despite the broad right of distribution conveyed by the GPL, enforcement of copyright is generally not possible for distributors: only the copyright holder or someone having assignment of the copyright can enforce the license. If there are multiple authors of a copyrighted work, successful enforcement depends on having the cooperation of all authors.
>
> In order to make sure that all of our copyrights can meet the recordkeeping and other requirements of registration, and in order to be able to enforce the GPL most effectively, FSF requires that each author of code incorporated in FSF projects provide a copyright assignment, and, where appropriate, a disclaimer of any work-for-hire ownership claims by the programmer's employer. That way we can be sure that all the code in FSF projects is free code, whose freedom we can most effectively protect, and therefore on which other developers can completely rely.

Please get in touch with the Gnuastro maintainer (currently Mohammad Akhlaghi, akhlaghi -at- gnu -dot- org) to follow the procedures. It is possible to do this for each change (good for for a single contribution), and also more generally for all the changes/additions you do in the future within Gnuastro. So if you have already assigned the copyright of your work on another GNU software to the FSF, it should be done again for Gnuastro. The FSF has staff working on these legal issues and the maintainer will get you in touch with them to do the paperwork. The maintainer will just be informed in the end so your contributions can be merged within the Gnuastro source code.

Gnuastro will gratefully acknowledge (see Section 1.11 [Acknowledgments], page 12) all the people who have assigned their copyright to the FSF and have thus helped to guarantee the freedom and reliability of Gnuastro. The Free Software Foundation will also acknowledge your copyright contributions in the Free Software Supporter: `https://www.fsf.org/free-software-supporter` which will circulate to a very large community (104,444 people in April 2016). See the archives for some examples and subscribe to receive interesting updates. The very active code contributors (or developers) will also be recognized as project members on the Gnuastro project webpage (see Section 11.3 [Gnuastro project webpage], page 238) and can be given a `gnu.org` email address. So your very valuable contribution and copyright assignment will not be forgotten and is highly appreciated by a very large community. If you are reluctant to sign an assignment, a disclaimer is also acceptable.

Do I need a disclaimer from my university or employer? It depends on the contract with your university or employer. From the FSF's `/gd/gnuorg/conditions.text`: "If you are employed to do programming, or have made an agreement with your employer that says it owns programs you write, we need a signed piece of paper from your employer disclaiming rights to" Gnuastro. The FSF's copyright clerk will kindly help you decide, please consult the following email address: "assign -at- gnu -dot- org".

11.11.2 Commit guidelines

To be able to cleanly integrate your work with the other developers, **never commit on the `master` branch** (see Section 11.11.3 [Production workflow], page 253, for a complete discussion and Section 11.11.4 [Forking tutorial], page 254, for a cookbook example). In short, leave `master` only for changes you fetch, or pull from the official repository (see Section 3.2.2.2 [Synchronizing], page 33).

In the Gnuastro commit messages, we strive to follow these standards. Note that in the early phases of Gnuastro's development, we are experimenting and so if you notice earlier commits don't satisfy some of the guidelines below, it is because they predate that guideline.

Commit title

The commits have to start with one short descriptive title. The title is separated from the body with one blank line. Run `git log` to see some of the most recent commit messages as an example. In general, the title should satisfy the following conditions:

- It is best for the title to be short, about 60 (or even 50) characters. Most emulated command-line terminals are about 80 characters wide. However, we should also allow for the commit hashes which are printed in `git log --oneline`, and also branch names or the graph structure outputs of `git log` which are also commonly used.
- The title should not finish with any full-stops or periods ('`.`').

Commit body

The body of the commit message is separated from the title by one empty line. Recall that anyone who has subscribed to `gnuastro-commits` mailing list will get the commit in their email after it has been pushed to `master`. People will also read them when they synchronize with the main Gnuastro repository (see Section 3.2.2.2 [Synchronizing], page 33). Finally, the commit messages will later be used to update the `NEWS` file on each release. Therefore the commit message body plays a very important role in the development of Gnuastro, so please adhere to the following guidelines.

- The body should be very descriptive. Start the commit message body by explaining what changes your commit makes from a user's perspective (added, changed, or removed options, or arguments to programs or libraries, or modified algorithms, or new installation step, or etc).
- Try to explain the committed contents as best as you can. Recall that the readers of your commit message do not necessarily have your current background. After some time you will also forget the context, so this request is not just for others[16]. Therefore be very descriptive and explain as much as possible: what the bug/task was, justify the way you fixed it and discuss other possible solutions that you might not have included. For the last item, it is best to discuss them thoroughly as comments in the appropriate section of the code, but only give a short summary in the commit message. Note that all added and removed source code lines will also be circulated in the `gnuastro-commits` mailing list.

[16] `http://catb.org/esr/writings/unix-koans/prodigy.html`

- Like all other Gnuastro's text files, the lines in the commit body should not be longer than 75 characters, see Section 11.5 [Coding conventions], page 240. This is to ensure that on standard terminal emulators (with 80 character width), the `git log` output can be cleanly displayed (note that the commit message is indented in the output of `git log`). If you use Emacs, Gnuastro's `.dir-locals.el` file will ensure that your commits satisfy this condition (using `M-q`).
- When the commit is related to a task or a bug, please include the respective ID (in the format of `bug/task #ID`, note the space) in the commit message (from Section 11.3 [Gnuastro project webpage], page 238) for interested people to be able to followup the discussion that took place there. If the commit fixes a bug or finishes a task, the recommended way is to add a line after the body with '`This fixes bug #ID.`', or '`This finishes task #ID.`'. Don't assume that the reader has internet access to check the bug's full description when reading the commit message, so give a short introduction too.

11.11.3 Production workflow

Fortunately 'Pro Git' has done a wonderful job in explaining the different workflows in Chapter 5[17] and in particular the "Integration-Manager Workflow" explained there. The implementation of this workflow is nicely explained in Section 5.2[18] under "Forked-Public-Project". We have also prepared a short tutorial in Section 11.11.4 [Forking tutorial], page 254. Anything on the master branch should always be tested and ready to be built and used. As described in 'Pro Git', there are two methods for you to contribute to Gnuastro in the Integration-Manager Workflow:

1. You can send commit patches by email as fully explained in 'Pro Git'. This is good for your first few contributions. Just note that raw patches (containing only the diff) do not have any meta-data (author name, date and etc). Therefore will not allow us to fully acknowledge your contributions as an author in Gnuastro: in the `AUTHORS` file and at the start of the PDF book. These author lists are created automatically from the version controlled source.

   To receive full acknowledgement when submitting a patch, is thus advised to use Git's `format-patch` tool. See Pro Git's Public project over email (`https://git-scm.com/book/en/v2/Distributed-Git-Contributing-to-a-Project#Public-Project-over-Email`) section for a nice explanation. If you would like to get more heavily involved in Gnuastro's development, then you can try the next solution.
2. You can have your own forked copy of Gnuastro on any hosting site you like (GitHub, GitLab, BitBucket, or etc) and inform us when your changes are ready so we merge them in Gnuastro. This is more suited for people who commonly contribute to the code (see Section 11.11.4 [Forking tutorial], page 254).

In both cases, your commits (with your name and information) will be preserved and your contributions will thus be fully recorded in the history of Gnuastro and in the `AUTHORS` file and this book (second page in the PDF format) once they have been incorporated into

[17] `http://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows`

[18] `http://git-scm.com/book/en/v2/Distributed-Git-Contributing-to-a-Project`

the official repository. Needless to say that in such cases, be sure to follow the bug or task trackers (or subscribe to the `gnuastro-devel` mailing list) and contact us before hand so you don't do something that someone else is already working on. In that case, you can get in touch with them and help the job go on faster, see Section 11.3 [Gnuastro project webpage], page 238. This workflow is currently mostly borrowed from the general recommendations of Git[19] and GitHub. But since Gnuastro is under heavy development currently, these might change and evolve to better suite our needs.

11.11.4 Forking tutorial

This is a tutorial on the second suggested method that you can submit your modifications in Gnuastro (see Section 11.11.3 [Production workflow], page 253), which is commonly known as forking. Let's assume you want to start contributing to Gnuastro and you would like to use GitLab[20] to host your work remotely and share it with other Gnuastro developers once you are ready. You can create an empty repository on your hosting service webpage. If this is your first hosted repository on the webpage, you also have to upload your public SSH key[21] for the `git push` command below to work. Here we'll assume you use the name `janedoe` to refer to yourself everywhere and that you choose `gnuastro-janedoe` as the name of your Gnuastro fork. Any online hosting service will give you an address (similar to '`git@gitlab.com:`' below) of the empty repository you have created using their webpage, use that address in the third line below.

```
$ git clone git://git.sv.gnu.org/gnuastro.git
$ cd gnuastro
$ git remote add janedoe git@gitlab.com:janedoe/gnuastro-janedoe.git
$ git push janedoe master
```

The full Gnuastro history is now pushed onto your hosting service and the `janedoe` remote is now also following your `master` branch. If you run `git remote show REMOTENAME` for the `origin` and `janedoe` remotes, you will see their difference: the first has pull access and the second doesn't. This nicely summarizes the main idea behind this workflow: you push to your remote repository, we pull from it and merge it into `master`, then you finalize it by pulling from the main repository.

To test (compile) your changes during your work, you will need to bootstrap the version controlled source, see Section 3.2.2.1 [Bootstrapping], page 32, for a full description. The process above is only necessary for your first time setup, you don't need to repeat it. However, please repeat the steps below for each independent issue you intend to work on.

Let's assume you have found a bug in `lib/statistics.c`'s median calculating function. You announce it (see Section 1.7 [Report a bug], page 9) so everyone knows you are working on it. However, if it is a very simple, clear/obvious and straightforward bug to fix you can skip this initial announcement. With the commands below, you make a branch, checkout to it, correct the bug, check if it is indeed fixed, add it to the staging area, commit it to the new branch and push it to your hosting service. But before all of them, make sure that you are on the `master` branch and that your `master` branch is up to date with the main Gnuastro repo with the first two commands.

[19] `https://github.com/git/git/blob/master/Documentation/SubmittingPatches`

[20] See `https://www.gnu.org/software/repo-criteria-evaluation.html` for an evaluation of the major existing repositories.

[21] For example see this explanation provided by GitLab: `http://docs.gitlab.com/ce/ssh/README.html`.

```
$ git checkout master
$ git pull
$ git checkout -b bug-median-stats      # Choose a descriptive name
$ emacs lib/statistics.c
$                                       # do your checks here
$ git add lib/statistics.c
$ git commit
$ git push janedoe bug-median-stats
```

Your new branch is now on your hosted repository. Through the respective tacker on Savannah (see Section 11.3 [Gnuastro project webpage], page 238) you can then let the other developers know that your `bug-median-stats` branch is ready. They will pull your work, test it themselves and if it is ready to be merged into the main Gnuastro history, they will merge it into the `master` branch. After that is done, you can simply checkout your local `master` branch and pull all the changes from the main repo. After the pull you can run '`git log`' as shown below, to see how `bug-median-stats` is merged with master. So you can push all the changes to your hosted repository and delete the branches:

```
$ git checkout master
$ git pull
$ git log --oneline --graph --decorate --all
$ git push janedoe master
$ git branch -d bug-median-stats                # delete local branch
$ git push janedoe --delete bug-median-stats    # delete remote branch
```

Just as a reminder, always keep your work on each issue in a separate local and remote branch so work can progress on them independently. After you make your announcement, other people might contribute to the branch before merging it in to `master`, so this is very important. Also before starting each issue branch from `master`, be sure to run `git pull` in `master` as shown above, to start your branch (work) from the most recent history point and thus simplify the final merging of your work.

Appendix A Gnuastro programs list

GNU Astronomy Utilities 0.2, contains the following programs. They are sorted in alphabetical order and followed by their version number. A short description is provided for each program which starts with the executable names in `thisfont` followed by a link to the respective section in parenthesis, see Section 1.4 [Naming convention], page 5. Throughout this book, they are ordered based on their context, please see the book contents for contextual ordering.

Arithmetic
: (`astarithmetic`, see Section 6.2 [Arithmetic], page 83) For arithmetic operations on multiple (theoretically unlimited) number of datasets (images). It has a large and growing set of arithmetic, mathematical, and even statistical operators (for example `+`, `-`, `*`, `/`, `sqrt`, `log`, `min`, `average`, `median`).

ConvertType
: (`astconvertt`, see Section 5.2 [ConvertType], page 65) Convert astronomical data files (FITS or IMH) to and from several other standard image and data formats, for example TXT, JPEG, EPS or PDF.

Convolve
: (`astconvolve`, see Section 6.3 [Convolve], page 88) Convolve (blur or smooth) data with a given kernel in spatial and frequency domain on multiple threads. Convolve can also do de-convolution to find the appropriate kernel to PSF-match two images. Convolve (blur or smooth) data with a given kernel.

CosmicCalculator
: (`astconvolve`, see Section 9.1 [CosmicCalculator], page 182) Do cosmological calculations, for example the luminosity distance, distance modulus, comoving volume and many more.

Header
: (`astheader`, see Section 5.1 [Header], page 62) Print and manipulate the header data of a FITS file.

ImageCrop
: (`astimgcrop`, see Section 6.1 [ImageCrop], page 75) Crop region(s) from an image and stitch several images if necessary. Inputs can be in pixel coordinates or world coordinates.

ImageStatistics
: (`astimgstat`, see Section 7.1 [ImageStatistics], page 128) Get pixel statistics and save histogram and cumulative frequency plots.

ImageWarp
: (`astimgwarp`, see Section 6.4 [ImageWarp], page 109) Warp image to new pixel grid. Any projective transformation or Homography can be applied to the input images.

MakeCatalog
: (`astmkcatalog`, see Section 7.3 [MakeCatalog], page 145) Make catalog of labeled image (output of NoiseChisel). The catalogs are highly customizable and adding new calculations/columns is very streightforward.

MakeNoise
: (`astmknoise`, see Section 8.2 [MakeNoise], page 176) Make (add) noise to an image, with a large set of random number generators and any seed.

MakeProfiles
: (`astmkprof`, see Section 8.1 [MakeProfiles], page 162) Make mock 2D profiles in an image. The central regions of radial profiles are made with a configurable 2D Monte Carlo integration. It can also build the profiles on an over-sampled image.

NoiseChisel
: (`astnoisechisel`, see Section 7.2 [NoiseChisel], page 136) Detect and segment signal in noise. It uses a technique to detect very faint and diffuse, irregularly shaped signal in noise (galaxies in the sky), using thresholds that are below the Sky value, see arXiv:1505.01664 (`http://arxiv.org/abs/1505.01664`).

SubtractSky
: (`astsubtractsky`, Section 6.5 [SubtractSky], page 116) Find and subtract sky value by comparing the mode and median on a mesh grid.

Table
: (`asttable`, Section 5.3 [Table], page 71) Convert FITS binary and ASCII tables into other such tables, print them on the command-line, save them in a plain text file, or get the FITS table information.

Appendix B Other useful software

In this appendix the installation of programs and libraries that are not direct Gnuastro dependencies are discussed. However they can be useful for working with Gnuastro.

B.1 SAO ds9

SAO ds9[1] is not a requirement of Gnuastro, it is a FITS image viewer. So to check your inputs and outputs, it is one of the best options. Like the other packages, it might already be available in your distribution's repositories. It is already pre-compiled in the download section of its webpage. Once you download it you can unpack and install (move it to a system recognized directory) with the following commands (`x.x.x` is the version number):

```
$ tar xf ds9.linux64.x.x.x.tar.gz
$ sudo mv ds9 /usr/local/bin
```

Once you run it, there might be a complaint about the Xss library, which you can find in your distribution package management system. You might also get an `XPA` related error. In this case, you have to add the following line to your `~/.bashrc` and `~/.profile` file (you will have to log out and back in again for the latter):

```
export XPA_METHOD=local
```

B.1.1 Viewing multiextension FITS images

The FITS definition allows for multiple extensions inside a FITS file, each extension can have a completely independent data set inside of it. If you ordinarily open a multi-extension FITS file with SAO ds9, for example by double clicking on the file or running `$ds9 foo.fits`, SAO ds9 will only show you the first extension. To be able to switch between the extensions you have to follow these menus in the SAO ds9 window: File→Open Other→Open Multi Ext Cube and then choose the Multi extension FITS file in your computer's file structure.

The method above is a little tedious to do every time you want view a multi-extension FITS file. Fortunately SAO ds9 also provides options that you can use to specify a particular behavior. One of those options is `-mecube` which opens a FITS image as a multi-extension data cube. So on the command-line, if you run `$ds9 -mecube foo.fits` a small window will also be opened, which allows you to switch between the image extensions that `foo.fits` might have. If `foo.fits` only consists of one extension, then SAO ds9 will open as usual.

Just to avoid confusion, note that SAO ds9 does not follow the GNU style of separating long and short options as explained in Section 4.1.1 [Arguments and options], page 45. In the GNU style, this 'long' option should have been called like `--mecube`, but SAO ds9 does follow those conventions and has its own.

It is really convenient if you set ds9 to always run with the `-mecube` option on your graphical display. On GNOME 3 (the most popular graphic user interface for GNU/Linux systems) you can do this by taking the following steps:

- Open your favorite text editor and put the following text in a file that ends with `.desktop`, for example `saods9.desktop`. The file is very descriptive.

  ```
  [Desktop Entry]
  ```

[1] `http://ds9.si.edu/`

```
Type=Application
Version=1.0
Name=SAO ds9
Comment=View FITS images
Exec=ds9 -mecube %f
Terminal=false
Categories=Graphic;FITS;
```

- Copy this file into your local (user) applications directory:

  ```
  $ cp saods9.desktop ~/.local/share/applications/
  ```

  In case you don't have the directory, you can make it yourself:

  ```
  $ mkdir -p ~/.local/share/applications/
  ```

- The steps above will add SAO ds9 as one of your applications. To make it default for every time you click on a FITS file. Right click on a FITS file and select "Open With", then go into "Other Application..." and choose "SAO ds9".

In case you are using GNOME 2 you can take the following steps: right click on a FITS file and choose Properties→Open With→Add button. A list of applications will show up, ds9 might already be present in the list, but don't choose it because it will run with no options. Below the list is an option "Use a custom command". Click on it and write the following command: `ds9 -mecube` in the box and click "Add". Then finally choose the command you just added as the default and click the "Close" button.

B.2 PGPLOT

PGPLOT is a package for making plots in C. It is not directly needed by Gnuastro, but can be used by WCSLIB, see Section 3.1.1.3 [WCSLIB], page 26. As explained in Section 3.1.1.3 [WCSLIB], page 26, you can install WCSLIB without it too. It is very old (the most recent version was released early 2001!), but remains one of the main packages for plotting directly in C. WCSLIB uses this package to make plots if you want it to make plots. If you are interested you can also use it for your own purposes.

If you want your plotting codes in between your C program, PGPLOT is currently one of your best options. The recommended alternative to this method is to get the raw data for the plots in text files and input them into any of the various more modern and capable plotting tools separately, for example the Matplotlib library in Python or PGFplots in LaTeX. This will also significantly help code readability. Let's get back to PGPLOT for the sake of WCSLIB. Installing it is a little tricky (mainly because it is so old!).

You can download the most recent version from the FTP link in its webpage[2]. You can unpack it with the `tar xf` command. Let's assume the directory you have unpacked it to is `PGPLOT`, most probably it is: `/home/username/Downloads/pgplot/`. open the `drivers.list` file:

```
$ gedit drivers.list
```

Remove the `!` for the following lines and save the file in the end:

```
PSDRIV 1 /PS
PSDRIV 2 /VPS
```

[2] `http://www.astro.caltech.edu/~tjp/pgplot/`

```
PSDRIV 3 /CPS
PSDRIV 4 /VCPS
XWDRIV 1 /XWINDOW
XWDRIV 2 /XSERVE
```

Don't choose GIF or VGIF, there is a problem in their codes.

Open the `PGPLOT/sys_linux/g77_gcc.conf` file:

```
$ gedit PGPLOT/sys_linux/g77_gcc.conf
```

change the line saying: `FCOMPL="g77"` to `FCOMPL="gfortran"`, and save it. This is a very important step during the compilation of the code if you are in GNU/Linux. You now have to create a folder in `/usr/local`, don't forget to replace `PGPLOT` with your unpacked address:

```
$ su
# mkdir /usr/local/pgplot
# cd /usr/local/pgplot
# cp PGPLOT/drivers.list ./
```

To make the Makefile, type the following command:

```
# PGPLOT/makemake PGPLOT linux g77_gcc
```

It should finish by saying: `Determining object file dependencies`. You have done the hard part! The rest is easy: run these three commands in order:

```
# make
# make clean
# make cpg
```

Finally you have to place the position of this directory you just made into the `LD_LIBRARY_PATH` environment variable and define the environment variable `PGPLOT_DIR`. To do that, you have to edit your `.bashrc` file:

```
$ cd ~
$ gedit .bashrc
```

Copy these lines into the text editor and save it:

```
PGPLOT_DIR="/usr/local/pgplot/"; export PGPLOT_DIR
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/pgplot/
export LD_LIBRARY_PATH
```

You need to log out and log back in again so these definitions take effect. After you logged back in, you want to see the result of all this labor, right? Tim Pearson has done that for you, create a temporary folder in your home directory and copy all the demonstration files in it:

```
$ cd ~
$ mkdir temp
$ cd temp
$ cp /usr/local/pgplot/pgdemo* ./
$ ls
```

You will see a lot of pgdemoXX files, where XX is a number. In order to execute them type the following command and drink your coffee while looking at all the beautiful plots! You are now ready to create your own.

```
$ ./pgdemoXX
```

Appendix C GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any,

be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index: Macros, structures and functions

All Gnuastro library's exported macros start with `GAL_`, and its exported structures and functions start with `gal_`. This abbreviation stands for *G*NU *A*stronomy *L*ibrary. The next element in the name is the name of the header which declares or defines them, so to use the `gal_array_fset_const` function, you have to `#include <gnuastro/array.h>`. See Section 10.2 [Gnuastro library], page 197, for more. The `pthread_barrier` constructs are our implementation and are only available on systems that don't have them, see Section 10.2.11.1 [Implementation of `pthread_barrier`], page 231.

G

P

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z