# ASTR4004/ASTR8004
# Astronomical Computing

Christoph Federrath

2025

## Contents

# 1 Prerequisites

## 1.1 Computing environment and shell

1. Before we can get started with Astronomical Computing, we need a computer with certain programs installed. The most important such program is the 'shell'. We will be using the Bourne Again Shell (Bash) in this course. First of all, you will need a computer with a shell program installed. This is always the case for Unix/Linux and Mac operating systems. Windows also provides a command line tool, but you will have to install a Unix-type shell. For Windows users there is a program called 'cygwin', which provides all the advanced functionality of a Unix shell, including X11 forwarding (we'll get to what that is). However, newer Windows versions also support a Linux subsystem, which may be the best choice if you want to use Windows. This is a virtual machine that simulates a Unix/Linux environment. For more performance, however, you can also consider a dual-boot setup, where you would have Windows on one of your hard drive partitions and Unix/Linux on the other. That way, Windows and Linux are more separated. For Mac users, I recommend to install 'iTerm2'. The Mac operating system (OS) is a Unix-type operating system, so you won't need to go through as much setup hustle as you would on a Windows machine.

2. Now make sure you have Bash installed on your computer. Open a terminal program (e.g., 'iTerm' on the Mac, and 'terminal' or "cygwin' on Windows) and type:
```
> echo $SHELL
```
This should give you as output something like `/bin/bash`. The last part of the path is the file `bash`, which is the Bash program that starts a Bash shell.

3. Bash is a shell program designed to listen to your commands, do what you tell it to, as well as to browse and modify files and directories on your computer. Bash is a text-based program, so it may seem limiting at first, but we will see that it is very powerful and provides you with full control over advanced tasks, such as scripting (non-interactive Bash mode) or connecting to other computers in the network.

4. Now make yourself familiar with the most basic functions of Bash. I recommend to go through the Bash guide (`http://guide.bash.academy`) or a similar online tutorial on Bash.

## 1.2 Package manager

1. Bash comes with a set of basic commands and some important standard programs will already be installed on your system. However, some other useful programs may not yet be installed, so we have to find a way to add them to your system.

2. In order to install additional programs, it is best to use so called 'package managers'. For Mac users, I recommend to install 'macports' (`https://www.macports.org`), which is a good package manager. Simply go to `https://www.macports.org/install.php` and follow the instructions to install macports. You will need to install Xcode and do a couple of further steps explained there.

3. For Linux users, let's assume you already know what you are doing :) ...and for Windows users the situation is basically hopeless :o ...oh well, you can install additional programs with the 'cygwin' package manager or the Linux subsystem package manager, depending on what type of Linux distribution is supported by the Windows subsystem for Linux (WSL). The default Linux type for WSL is Ubuntu.

4. Now you can install additional programs with your package manager, such as 'gnuplot'.

## 1.3 Window manager

If you are using a Mac, you will also need to install XQuartz from `https://www.xquartz.org`.

Note for Windows 10 users: if you are a Windows 10 user, you install Bash (`https://itsfoss.com/install-bash-on-windows/`) and you have to install the Xming X Server for Windows. You might also have to set `export DISPLAY=localhost:0.0` in your SHELL environment (e.g., add that to your `.bashrc`). Similar may apply to Windows 11. Some further information is located here: `https://wiki.iihe.ac.be/Use_X11_forwarding_with_WSL`.

Also note that if you use the Bash on Windows 10/11, there are some issues with file permissions, i.e., new files are usually given read/write/exec permissions not only for the user, but also for the group and everyone else. This can cause problems when configuring secure shell (in the following lectures) to connect to other computers/servers. You can fix the file permissions by hand (using `chmod go-rwx [file]`) in that case.

## 1.4 Other required/useful programs

1. First you need an editor, a program to open, modify, save and close files. I recommend to install 'vscode' (fancy; need to download and install by hand, but is very powerful) and 'emacs' (old school, simple and also very powerful). Using macports on the Mac (see previous section), you would do:
   ```
   > sudo port install emacs
   ```
   This will install emacs. You can start emacs from the shell, simply by typing `emacs`. You can exit with Ctrl-X-Ctrl-C. Have a look at the emacs manual to learn about basic commands to control emacs (such as open/close/save files, and exit, ...there is a lot more that can be done with emacs, but we will only use the most basic functions here).

2. Two other useful programs that we will need for some of the later lectures are 'rsync' and 'gnuplot'. The first one is a program to copy/synchronise files on your and other computers and the second one is a plotting package. If they are not yet installed, please install them with your package manager. See if they are installed by typing:

`> rsync`
which should bring up a list of command line options for rsync and
`> gnuplot`
which should start gnuplot. You can exit gnuplot by typing `gnuplot> exit`.

3. You can install more programs as required. For example to search for a program by name, in macports, you can type:
   `> sudo port search [name_of_program]`
   to see if that program is available for installation.

4. A note on X11 (so you can use windows and graphics on your own as well as on remote computers), the simplest is to install gnuplot
   `> sudo port install gnuplot`
   which comes with the required xorg libraries.

## 1.5 Word of caution

*If you need more information, please ask me and/or use the power of searching the internet to find some answers. But do not necessarily trust either of those :)*

*CAUTION*: be careful with copying-and-pasting code or shell commands from this PDF document into the shell or into another document (e.g., a script opened in an editor). You may not be getting what you are seeing on the screen, for example, depending on the exact editor, shell, and PDF viewer, the special characters '(', ')', ' ' ', '%' can all cause problems when performing copy-paste operations. This is because of potential differences in character encodings between the PDF document and the shell or script editor.

In general, you can use `man [command]` to bring up the manual page of a program/command to learn about possible command-line options for shell program(s), or often easier is to search the internet for answers/solutions; especially if you get unexpected errors or warnings. Usually, someone else with a similar or the same configuration of your computer/script/shell/code will have encountered a similar problem/issue and you primarily have to learn how to search for the right answer on the internet.

# 2 Remote computing

## 2.1 Setting up remote access to ANU services

1. Most of the MSO servers and also some ANU services are only accessible from *inside* the ANU network. This means that they normally cannot be accessed and used when you are at home. However, there is a service called 'VPN', which allows you to setup a secure connection, such that it looks to the ANU servers as if you were actually inside the Uni network, although you might be at home or even overseas.

2. Download the VPN server for your computer from the ANU VPN website. This is currently located here: `https://services.anu.edu.au/information-technology/login-access/remote-access`, called 'GlobalProtect' (I feel so much safer already...).

3. Follow the instructions on that web page to install the client program. This will be useful for later also, because you can then connect to MSO servers and ANU services from outside the University network. However, this may not give you immediate access to MSO computing services, because ANU IT has to register access rights to MSO for your

specific Uni account. Therefore, please fill submit the completed RSAA account form asap.

## 2.2 Customising the Bash shell environment

Working with the shell may at first seem a little limiting. However, if you start getting familiar with it, you can do lots more powerful things than the usual click-click-click stuff you might be used to when working on a computer. For example, you can do automation, customisation, create new commands, etc. You can also very easily connect to other computers as we will see shortly.

Before we begin, I recommend to install Visual Studio Code (`vscode`), which is a powerful text editor. This is particularly useful, because if you are on Windows and you use the Windows text editor, then each line in a file will normally be ended with the CRLF (Carriage Return and Line Feed) line-termination when you press Enter, while Unix-based systems use the LF (Line Feed) line-termination. If the wrong (CRLF) character encoding is used on Unix/Mac, then none of your scripts will work as intended.

1. First, let's change the Bash console prompt. Add to your `.bashrc` in your `$HOME` directory the line:
   ```
   export PS1='\u@\h:\w>'
   ```
   Btw, on Mac OS, the default shell is zsh and the `.bashrc` equivalent is called `.zshrc`. Then changing the prompt works via `export PS1='%n@%m:%~>'`

2. Now make sure that the following line is also present in your `.bashrc`:
   ```
   test -s ~/.alias && . ~/.alias
   ```
   This ensures that the content of `.alias` is read and added to the Bash environment. Can you explain the syntax and what's going on here in this one-liner?

3. We can now modify (or create, if it does not exist yet) the `.alias` and add some useful shell aliases that allow you to access commands more quickly. A common command would be listing an extended, more detailed version of `ls`, containing the file sizes, last modification date of the files, and file permissions and ownership. Please add the following lines to `.alias`:
   ```
   alias ls='ls -G $OPTIONS'
   alias ll='ls -Glh $OPTIONS'
   ```
   The first one should enable colourised output via the `-G` option, which applies to Mac OS (while the option `--color=auto` applies to Unix OS). Use `man ls` to see the manual page for the specific version and implementation of `ls` installed on the system that you are running `ls`.

   Note that `$OPTIONS` is there to take additional options. Basically, we have now overloaded the `ls` command with a colourised version. The second line defines a new command alias called `ll`, which essentially calls `ls` with the additional options `-l` and `-h` for long-listing (more details) and human-readable file size output, respectively.

   Try and understand the listing of files and directories via `ll`. When you run this from your home directory, what do the different columns mean, in particular the first column, which is encoding the file/directory permissions?

4. Now let's add a quick alias for your favourite editor, so you can quickly launch it from the shell, for example `emacs`:
   ```
   alias e='emacs $OPTIONS'
   ```
   Thus next time, you can very quickly open emacs by just typing `e -nw [file]`. Note

that the -nw option refers to 'no window', meaning that emacs will be started directly in shell mode instead of graphical-user-interface (GUI) mode. This is very useful if we want to modify a file on a remote computer, where we do not have a fast network connection for graphical interactions. The text-based editor mode is much faster and is usually sufficient for most tasks. If you don't already have an efficient editor that you know how to use remotely, I recommend to learn how to use emacs and common short-cuts by searching the manual pages of emacs on the internet.

5. Make sure to restart Bash (or open a new Bash shell) so that your changes to .bashrc and .alias will take effect. Another way of activating the changes is to call source .bashrc.

6. If this does not work, you have to check your .profile or .bash_profile. There should be a call to .bashrc from either of those, similar to:
```
if [ "$BASH" ]; then
 if [ -f ~/.bashrc ]; then
  . ~/.bashrc
 fi
fi
```

## 2.3 Connecting to another computer using the shell

1. Connecting to another computer (also called remote host) in the local network or sometimes even the internet is easy with Bash. First, let's use the main program to do this, which is ssh ('Secure Shell'). Type
```
> ssh [your_mso_username]@motley.anu.edu.au
```
This connects you to the host motley.anu.edu.au at MSO.

2. You may find that your shell prompt on motley.anu.edu.au is different from the one we just set up on your local computer. This is because every host has its own Bash environment. Follow the procedures above that we used to set up our local environment via the files .bashrc and .alias to generate the same basic Bash environment on your remote account at RSAA/MSO. After making those modifications, log out (using command exit), and then login again. You should now have the updated environment on motley.anu.edu.au, and in fact on any other MSO server, because your home directory is mounted on every server; in other words you could also connect to motley.anu.edu.au or avatar.anu.edu.au. In other words, your home space is shared by all servers at RSAA/MSO, so you don't have to set this up again when you connect to a different server at MSO, because each server at RSAA/MSO will read the same files and start up your Bash environment in the same (or at least similar) way, because your $HOME will be mounted on every MSO server. So you also have access to all your files on every MSO server.

3. Make sure you have a window manager installed (on Mac OS, you can install XQuartz from https://www.xquartz.org). Now let's try to plot something with gnuplot:
```
> gnuplot
> plot sin(x)
```
The first command starts gnuplot, a handy program to plot data in ASCII text files or analytic functions real quick. In gnuplot, plot sin(x) should plot a sin function. However, if you strictly followed this guide, you will get no plot, but instead an error message 'gnuplot: unable to open display, X11 aborted.' or something like that. The

problem is that we only connected to motley.anu.edu.au in text/shell mode, but in order to bring up a plot window that shows the plot, we have to ssh-connect in *graphics mode*—we have to enable what's called 'X11 forwarding'.

4. In order to achieve X11 forwarding, we have to logout and reconnect using the ssh option -X or -Y, in order to enable X11 forwarding. Let's try:
   ```
   > ssh -Y [your_mso_username]@motley.anu.edu.au
   ```
   ...and then try to plot the sin again with gnuplot. This should now bring up a new window with the sin(x) plot.

## 2.4 Customising ssh connections

1. Now we know how to connect to remote servers/hosts. However, the process and command for the connection is a bit lengthy, so in order to copy files and to connect more easily without having to specify your user name and the full remote hostname all the time, there is a neat way to customise ssh connections. On your local computer, change into your home directory: > cd
   Then, change into the directory .ssh/. Note that this is a hidden directory (which is why it starts with a dot), so ls would normally not show it, however, if you do ls -a, then all files and directories are shown, including hidden ones. In case this does not exist, create the directory with > mkdir ~/.ssh/.

2. So, now in directory .ssh/, see if you have a file called config. If not, then simply create it (or if it exists), modify (creation and modification is the same command if you are using emacs):
   ```
   > e -nw config
   ```
   Now let's add the following lines:
   ```
   Host motley
   Hostname motley.anu.edu.au
   User [your_mso_username]
   ```
   These three lines define a new host name alias called motley, which will make it a lot easier for you to connect to motley. You don't even have to remember your (possibly a bit cryptic) user name, because you'll see that this is automatically used now every time you connect or copy files to/from MSO servers.

3. On a Mac, you may also have to add the following to config:
   ```
   XAuthLocation /opt/local/bin/xauth
   ```
   in order to get x11 forwarding working.

4. Save and close config and now try to do the following:
   ```
   > ssh -Y motley
   ```
   This should connect you to motley directly. Note that we do not have to specify your user name or the full remote host name anymore; just motley.

5. Finally, define an alias called 'motley' in your local .alias with the ssh command above, including X11 forwarding to motley. This will allow you to simply type: > motley
   to start a connection to motley.anu.edu.au with your MSO user name from your local computer.

## 2.5 Setting up an ssh tunnel

1. If you are connected to the VPN (see Section 2.1 above), you can directly connect to motley.anu.edu.au. However, if you are outside the ANU network and you cannot

use the VPN client for some reason, there is still another possibility to connect to the MSO servers. This requires you to first connect to a specific server at MSO that is accessible from the outside world. This server is called 'msossh1.anu.edu.au'. **However, as of 2021, this is not an option anymore; ANU wants us to use the VPN :(** . . . Nevertheless, we will use this as a basic example for how to set up an ssh tunnel. For instance, some supercomputing centres will only allow access from specific IP addresses, in which case you might want to use your laptop computer (usually dynamic IP address assignment) to tunnel through that registered computer onto the supercomputer that sits behind a firewall.

2. Let's pretend `motley.anu.edu.au` might be protected and can only be reached from `msossh1.anu.edu.au`. In order to connect to `motley.anu.edu.au`, we simply connect to `msossh1.anu.edu.au` first and then `ssh` from `msossh1.anu.edu.au` to `motley.anu.edu.au`. However, in order to make our life easier, especially when copying files between remote computers, we can set up an `ssh` tunnel (in order to avoid the two-step process of first connecting/copying to `msossh1.anu.edu.au` and then to `motley.anu.edu.au`). Do this by adding/modifying the following lines in your `.ssh/config`:
```
Host motley
Hostname motley.anu.edu.au
User [your_mso_username]
ProxyCommand ssh -q -a -Y [your_mso_username]@msossh1.anu.edu.au nc %h %p
```
Save and close `.ssh/config`.

3. Now you should be able to connect to `motley.anu.edu.au` by using `> motley` directly from your local computer. It will probably ask for your password twice; the first time when it connects to `msossh1.anu.edu.au` and the second time when it connects from there to `motley.anu.edu.au`. So what this effectively does is that we tunnel through `msossh1` to `motley`.

4. As earlier, add a 'motley' alias to your `.alias` file as a shortcut for `ssh -Y motley`.

## 2.6 Setting up private and public key pairs to connect to remote hosts without using a password

In order to establish password-less connections to remote hosts, you can follow the instructions, e.g., on this webpage: `http://www.linuxproblem.org/art_9.html`. Note however, that this is at your own risk. It may be easier for hackers to get access to your account *and automatically to all servers where you copied your public key to for password-less access.* In any case, if you create a public/private key pair, never give your *private key* to anyone and make sure it cannot be read by others than you (strictly set user-only permissions via `chmod`) in your `~/.ssh/` directory, expect for the public key. You can give the public key to anyone, so they can give you access to their resources; for example, later, we will look into version control software and web services such as bitbucket or github. You can upload your public key there, so you can access your online repositories more easily.

## 2.7 Copying files/data across remote computers

1. With the changes to `.ssh/config` that we made earlier, it is now also very simple to copy files between your computer and the remote host. Let's copy the density PDF data file from one of the assignments into your home directory on motley.anu.edu.au. First, download the data file to your local computer (if you don't have it already) from `http://www.`

mso.anu.edu.au/~chfeder/teaching/astr_4004_8004/material/mM4_10048_pdfs/EXTREME_hdf5_plt_cnt_0050_dens.pdf_ln_data

2. Now copy the data file to `motley` using `scp` ('secure copy'):
   `> scp EXTREME_hdf5_plt_cnt_0050_dens.pdf_ln_data motley:`
   Note that you can use the tab key to auto-complete the rest of the awkward filename. If you just provided the initial few letters 'EX' and hit the tab key, the shell will automatically complete the rest of the filename. Note also that when you start `> scp` and then hit the tab key twice, you will be given a list of possible options for files in this directory. This is a very useful auto-completion function of the shell. Once you executed the `scp` command, you should see the file being transferred to motley. Note that the colon behind 'motley' refers to your home directory on motley. If you want to copy the file somewhere else, you just have to expand the path to the destination folder on motley after the colon.

## 2.8 Mounting remote file systems

1. If you are frequently accessing files on a remote host and would like to use them on your computer, you can use `sshfs` to mount a remote file system on your computer. This means you don't have to explicitly copy files every time, but they will simply be present in one of your local directories on your laptop and only if needed, will the files in there be copied via the network. Mounting a filesystem remotely can be useful if you want to use local tools (such as editors or visualisation software) on data files that are on a remote server. However, note that the communication can be pretty slow, depending on the speed of your network connection.

2. For example, if we want to mount your $HOME from the MSO network on your laptop, we can first create a directory in your local (on your laptop) computer, called 'mso_home':
   `> mkdir ~/mso_home`

3. Now we mount your home dir at MSO into this folder: `> sshfs mso: ~/mso_home/`

4. Be very careful when you move files in or out of this directory `~/mso_home`, because any modification of it locally on your laptop will be exactly reflected on your home dir in the MSO system. Moreover, the transfer of data goes via the network, so if you do this at your home it can take bandwidth from your internet connection, depending on the operations that you are doing in this directory; e.g., copying a 1 GB file in or out of `~/mso_home` will take time depending on your internet connection and will of course transfer that amount of data.

5. You can un-mount the directory (on Mac OS) via: `> diskutil unmount ~/mso_home/`

6. A very convenient way to connect to a remote server is to use `vscode`. You can install the 'Remote SSH' extension, which will give access to connecting to remote hosts. `vscode` can read your `.ssh/config` and you will be able to connect, seeing the entire accessible filesystem on the remote host.

## 2.9 Using rsync to create backups and to copy large data sets; use of tarballs

1. Download the tarball (a set of files bundled together in one compressed file) from http://www.mso.anu.edu.au/~chfeder/teaching/astr_4004_8004/material/mM4_10048_pdfs/EXTREME_pdfs.tar.gz

2. Create a new directory on your local computer called 'astro_comp' in your home directory:
   ```
   > mkdir astro_comp
   ```

3. Now move `EXTREME_pdfs.tar.gz` into this new directory:
   ```
   > mv EXTREME_pdfs.tar.gz astro_comp/
   ```
   ...and change into `astro_comp/`

4. Decompress and un-tar the tarball, using:
   ```
   > tar -zxvf EXTREME_pdfs.tar.gz
   ```
   This should generate 71 files. Do you remember how to check the total number of files in `astro_comp/`?

5. Now let's make a complete backup of the directory `astro_comp/` and all the files in it by sending a copy to motley.anu.edu.au. First change back into your home directory on your local machine. Then do:
   ```
   > rsync -av --stats astro_comp/ motley:astro_comp/
   ```
   This generates a complete 1:1 copy of your local directory `astro_comp/` in your `motley`'s home directory with the same directory name there. Please have a look at the status summary output when `rsync` returns from doing its job.

6. We could have used `scp` (see earlier) for pushing a copy of all the files to the remote server, but `rsync` can be used to backup data without having to transfer/copy all the files every time, but instead it only transfers changes (changed files). For instance, if you now run the same rsync command from above again, you will see that no files will be transferred (because nothing changed in your local copy). If, however, you were to modify one or more of the files in `astro_comp/` and you do the `rsync` again, only the modified files will be transferred. This is really useful also if you have many many big files to transfer to/from a computer or between different hosts, because if some of the files can't be transferred within a day or so and/or the ssh connection closes for some reason, at least rsync will continue from the same point where it stopped to synchronise the folders and files, instead of starting the copy process from scratch. BTW: an important part of astronomical computing is to make regular backups of your files :)

## 2.10 Use of `nohup` and `nice`

Here we learn how to launch a process on a remote host, and let it continue running in the background on the remote server, while we are logged out of the server after starting the process. This is useful if you have a script that is running for hours and you cannot keep an ssh connection open to the remote host for the whole time that the script/process is running for.

1. Make a Bash script called `nohup_script.sh` that prints the current time every second for the next hour. Use a bash loop and `date` to print the current date and time. Let the code pause for 1 second within each loop increment, by using `sleep 1`.

2. Start the script to see if it works. It should print the date and time to the screen every second. Note that you can stop the script by pressing Ctrl-C.

3. Now start the script, but redirect the stdout and stderr to a file `shell.out`. While the script is running in one shell, open another shell and look into the file. You should see that every second a new line is appended with the current date and time to the end of the file.

4. Start the same script, but this time with `nohup`. Simply place `nohup` in front of the command that starts the script and `&` after the command. `nohup` is a wrapper for any command that you want to start such that it does not hang up (`nohup` means 'no hang up') when you log out. The final `&` at the end of a command line is used to start that process in the background, which means that you get the shell back when you hit enter, but the program keeps running in the background.

5. Use `tail -f shell.out` to show the end of the output file. It also follows any changes to the file (the `-f` option). You should see how the file grows and how every second, the date and time is appended as a new line to the file.

6. Now log out of the MSO server. Since we have started the script with `nohup`, it should not hang up after we logged out, but instead will keep running on the server even though we are logged out. So, let's login again and see if the script kept on writing the time to the file while we were away.

7. Caution: if you start something with `nohup`, it will keep running unless it's killed by the user or an admin. We made the script so it would stop after one hour, but let's simply kill the script by hand right now. To achieve this, we use the `ps` command, which shows all running processes on the host. Let's first add a customised version of `ps` to our `.alias` file to make it an easy task to show all our own processes (there is a whole bunch of other processes that are also running simultaneously on the server, but we don't want to list all of them; just our own). Add the following alias to your aliases:
   ```
   alias myps='ps -elf |grep $USER'
   ```
   and log out and log back in for this change to take effect or simply `source .alias` in your home directory. Do you remember what the `|`-symbol (pipe) does?

8. Now do `> myps` in the shell. This should list your current active processes (see the pipe to `grep $USER`). Find the process ID that belongs to the running script that we started with `nohup` and kill the job using:
   ```
   > kill -9 [PID]
   ```
   where you have to replace '[PID]' with the process ID of the job to be killed. Alternatively, you can use
   ```
   > killall 'bash'
   ```
   which kills jobs based on their name. This will kill all your running bash scripts.

9. Note: You'd normally want to use `nohup` for jobs/scripts that take very long to run, for example over night or even longer and you don't want to keep an active shell open to the remote host while the script is running. This is when `nohup` is really useful. However, we should consider that other users might be running jobs or that a user could be physically sitting at that computer/host at the same time and trying to do some work. So a *nice* thing in this case is to use `nice`, which means that your job will only use CPU time (or compute power) when it is available. So in case the machine is under heavy load and running many processes at the same time, if you started your script with `nohup nice ./script.sh &`, then your job won't block the jobs of other users so much, because it waits for times when the machine is not under heavy work load.

# 3 Plotting and fitting with gnuplot, and making movies

## 3.1 Simple plots of data from ASCII text files

Gnuplot is a powerful plotting tools that can create plots from analytic functions and from tabulated data (e.g., from ASCII text files). Gnuplot is very useful for exploring data, plotting them in different ways, etc., even if that data is on a remote server. It can also be used to produce publication-quality figures, however, that requires tweaking a lot and making scripts, some of which we'll do here. Ultimately, IDL or python produce the 'nicest' plots with somewhat less effort, but making plots look 'perfect' always takes some fiddling with plot command options, etc., whichever programming language or package you are using.

1. Let's make some simple plots first. Change into your `$HOME/astro_comp/` on the MSO server (e.g., motley). Now look into `EXTREME_hdf5_plt_cnt_0050_dens.pdf_ln_data` with `more` or `less` or `cat` (the file should still be there from the previous lecture or you can download it again). You will see that there is a header in the file with 10 lines (showing the mean, standard deviation and other statistical moments of the distribution function), followed by an empty line and then 4 columns with data in them.

2. Now go back to the shell and start gnuplot. First, plot column 1 against column 3 in the data file. Use this command:
   ```
   gnuplot> plot "EXTREME_hdf5_plt_cnt_0050_dens.pdf_ln_data" using 1:3
   ```
   Note that you can also use auto-completion of the file name in gnuplot by hitting the tab key, just the same as in the shell. Ok, so this should show the density PDF in that file, which should look pretty close to a Gaussian.

3. Now replace "plot" with "p" and "using" with "u". This should do exactly the same as before, but is a bit more compact, i.e., no need to write out each gnuplot command, you can simply use the first letter in most cases and it will do the job.

4. Now plot the y-axis logarithmically. Do this with
   ```
   gnuplot> set log y
   ```
   and next enter the plot command from before again. Note that you can bring up the most recent last commands again, simply by using the up-arrow key; this should bring up the last few commands used. The same works in the Bash shell, which is very useful in case you made a mistake when typing the command, you can bring up the previous one instantly and correct only the bits of the command that didn't work or that you want to change—for example if you wanted to keep everything the same, but instead plot data from one of the other files in the directory.

5. Now plot on top of the previous plot, the respective data columns from file with number `*0060*`. Do this simply by adding to the end of the previous gnuplot command:
   ```
   , "EXTREME_hdf5_plt_cnt_0060_dens.pdf_ln_data" using 1:3
   ```
   This should now display two curves (or set of points) plotted on top of one another. You can add more lines/curves simply by appending more to the plot command. For example, if we wanted to plot now also a constant line at $y = 10^{-3}$, we'd simply add '`, 1e-3`' to the end of the previous gnuplot command line and we should see the horizontal line at $y = 10^{-3}$.

6. You can change the x and y axis labels with:
   ```
   gnuplot> set xlabel "log-density contrast"
   gnuplot> set ylabel "PDF"
   ```
   ...then plot again and see if the axis labels have changed.

7. Ok, so this is all fine, but it usually doesn't look very nice. Gnuplot is really good for making a quick plot of some data, but making it pretty needs a bit more tweaking. Now let's see what can be done to make nicer figures, generate gnuplot scripts and automatically write postscript figures to a file.

## 3.2 Fitting data with gnuplot

Gnuplot can be used to fit model functions to data.

1. Look at the data file `http://www.mso.anu.edu.au/~chfeder/teaching/astr_4004_8004/material/RSAA_publications.txt`, which contains RSAA's number of publications from year 2011 to 2018 and the cumulative number of citations extracted from `https://rsaa.anu.edu.au/research/publications`.

2. Plot the number of publications as a function of year. Then shift the data in the x-axis by -2011, such that year 2011 becomes 0.

3. Define a linear function $f(x) = ax + b$.

4. Now fit the RSAA publications data (note the shift in years, such that 2011 becomes 0):
```
fit f(x) "RSAA_publications.txt" u ($1-2011):2 via a,b
```

5. Plot the fit on top of the data.

## 3.3 Making scripts for gnuplot and generating postscript figures

1. Download the gnuplot script template `gnuplot.p` from `http://www.mso.anu.edu.au/~chfeder/teaching/astr_4004_8004/material/gnuplot.p` (or via the link on the course web page).

2. Look into the file and go through each line step-by-step and see if you can make sense of the commands. Similar to Bash scripts, gnuplot just goes through the script line-by-line and executes the commands as if you entered them one-by-one in the interactive gnuplot mode. The advantage of the script is obvious: you don't have to write it all to the gnuplot prompt and you can very quickly regenerate the same plot and/or make little changes easily. So it's usually a good idea to script a plot that you make, in order to come back to it any time later, in order to reproduce the same plot. Running the script is then achieved simply by loading the script in gnuplot:
```
gnuplot> load "gnuplot.p"
```
which executes all the commands in the script line-by-line.

3. In this particular script, we have already switched to a different output type (or terminal); in this case to 'postscript' (`set term post eps`), which generates `.eps` figures. Such figure files may contain vector graphics (much preferred, because scalable), but can also contain pixel graphics (for example maps). Note that the graphics editor `inkscape` is very useful, because it can handle vector graphics and can be used to modify figures, keeping the advantages of vector graphics. In contrast, the popular `gimp` graphics editor can only handle pixel graphics. So while you can load an image containing vector graphics in `gimp`, it will be automatically converted to a pixel graphics image and thus loses all the advantages of scalable vector graphics.

## 3.4 3D plots with gnuplot

1. Let's make a simple 3D plot of the previous figure in gnuplot. Use:
   ```
   splot "EXTREME_hdf5_plt_cnt_0050_dens.pdf_ln_data" u 1:2:3
   ```
   This shows the same density PDF as before, but now as a 3D plot. Note that you can turn the plot around by dragging it with your mouse.

2. Note that the 2nd column in the data file does not contain any useful data (it's all zeros), so it does not contain any information in the 2nd axis of the plot. Let's replace the 2nd column with the 4th column of the data file (which contains the cumulative distribution function):
   ```
   splot "EXTREME_hdf5_plt_cnt_0050_dens.pdf_ln_data" u 1:4:3
   ```
   This will now show a true 3D plot; i.e., the data in the 4th column of the file is now plotting along the 2nd axis of the 3D plot.

3. One can do more advanced things, for example changing the point types and adding a colour bar, e.g., by appending to the end of the previous line:
   ```
   splot ... u 1:4:3 with points palette pointsize 1 pointtype 6
   ```

## 3.5 Making movies from a time series of plots/images

1. First we have to write a gnuplot script that writes out `.png` figures for each of the PDF data files (the time sequence of the PDF) in the tarball from the last lecture.

2. We can use the basic gnuplot script from before, but in order to make png figures, let's change the gnuplot term to png:
   ```
   set term png size 800,480 enhanced font "Helvetica,14"
   ```

3. Then after some other formatting definitions and settings (e.g., line types, log, key position, and axes labels; see script template from earlier) we make a gnuplot loop like this:
   ```
   do for [i=20:90] {
       infile = sprintf('EXTREME_hdf5_plt_cnt_%04d_dens.pdf_ln_data',i)
       outfile = sprintf('frame_%04d.png',i-20)
       set output outfile
       p [-10:10] [1e-5:1] infile u 1:3 w lp ls 3 t sprintf('time=%04.0f',i)
       print outfile." created"
   }
   ```

4. When you run the script, it should generate a list of figures `frame_0000.png`, `frame_0001.png`, ..., `frame_0070.png` and print to the screen that those files were created.

5. Now we can use `ffmpeg` to make a movie out of the individual frames. You might want to install this on your own computer to be able to make the movie, or you can use it on motley. At the moment `ffmpeg` is located in `/pkg/linux/anaconda3/bin/`. You can add that directory to your PATH, by adding to your `.bashrc`:
   ```
   export PATH=$PATH:/pkg/linux/anaconda3/bin
   ```

6. Now that we have the still frames, we can make a movie. The simplest `ffmpeg` command to make a movie from a bunch of still frames is:
   ```
   > ffmpeg -i frame_%04d.png movie.mpeg
   ```
   where you have files `frame_0000.png`, `frame_0001.png`, etc., previously generated with gnuplot.

7. You can play the movie file with `ffplay`. However, it won't play well over the network, so I recommend to copy it to your local computer (`scp` or `rsync`; see earlier) and play it directly on your computer rather than in a window over the network.

8. So this is ok, but looks a bit pixelated. For a nicer movie, we have to increase the bit rate by adding the option `-b:v 5000k` to `ffmpeg`. This will greatly increase the bit rate and thus the quality of the movie output.

9. There are lots more advanced options in `ffmpeg`, for example cropping or extracting frames from movies and changing the encoder. It is one of the most powerful movie conversion/processing tools.

## 4 Extracting data from existing plots/graphs

1. Previously, we learned about plotting data from text files and/or functions in gnuplot. Here we now learn how to grab data from existing plots to digitize them in order to make our own plot of some published data from a graph. This can be very useful, because you might be working on a research project where you produce data that you want to directly compare with other existing data from the literature in the same plot frame. Instead of having to read the data from the plot by hand/eye and making a table by hand, there are nice tools that can make this process of extracting data from an existing plot much easier for you. Here we will focus on WebPlotDigitizer, which is such a tool.

2. First go to `https://automeris.io/WebPlotDigitizer/` and launch the App.

3. Go through the tutorial, which already has an example image of a plot loaded by default. You can extract data points manually or automatically (selected by colour and masks).

4. The first step is to calibrate the axes of the plot and then you are ready to extract data points.

5. Finally, the extracted data can be formatted as you wish and the extracted data pairs can be copied into a text file, which you can then use for further processing, e.g., for plotting with your own style and together with other data (e.g., from your own work), e.g., in gnuplot or IDL or python, etc.

6. Try it by uploading an image of a different plot/graph, calibrate the axes and see how you can extract data in manual and automatic mode. For example, you can upload the density PDF plot produced earlier, with a fitted Gaussian line drawn in a different colour. See if you can extract some data points of the fitted line and the underlying PDF data itself. Format them and copy them to a text file, which you can then read in gnuplot, so you can directly compare the original data and the extracted data.

## 5 The Interactive Data Language (IDL)

IDL was primarily developed by astronomers. Lots of code used in particular in the astro community is still IDL code, hence we're having a look at how its basics work. It is very similar to python, which is now getting more popular, but IDL was probably one of the first more widely used 'interpreted' (as opposed to 'compiled') programming languages.

## 5.1 IDL startup

1. Login to motley and make a new directory IDL/ in your home dir:
   ```
   > mkdir IDL
   ```
2. Download the IDL startup package prepared for you: http://www.mso.anu.edu.au/~chfeder/teaching/astr_4004_8004/material/IDL_startup_package.tar.gz
   and copy it to motley into the new IDL/ directory.
3. Unpack the tarball. This will create subdirectories ASTROLIB/, MPFIT/, and textoidl/, as well as three files: idlstartup, setcolors.pro, constants.pro.
4. ASTROLIB is a useful astronomy IDL library, MPFIT is an IDL non-linear fitting package, and textoidl is a Latex-to-IDL string conversion library that lets you use Latex syntax in IDL to make Greek letters, sub- and super-scripts and special symbols like you are used to in Latex.
5. The idlstartup file is useful, because it controls the way IDL starts up (similar to how .bashrc is run every time you start a new Bash session, idlstartup is run every time you start IDL). In our case, it defines paths and automatically runs the script constants.pro, which defines useful physical constants (the use of which, we will see below).
6. In order for IDL to know where to look for idlstartup, add this line to your .bashrc:
   ```
   export IDL_STARTUP=${HOME}/IDL/idlstartup
   ```
   This will make sure that idlstartup is executed every time you start IDL.

## 5.2 Simple IDL tasks

1. Now that we have set up the IDL environment, we can start IDL:
   ```
   > idl
   ```
   As for gnuplot, this will lead you to the IDL command line from which IDL is controlled.
2. First, let's make a simple calculation and print the result to the screen:
   ```
   idl> print, 1+1
   ```
3. We can also directly define variables, modify them and print their content:
   ```
   idl> a = 1+1
   idl> a = a*3
   idl> print, a
   ```
4. Finally, let's do some astro calculation involving units. Now that the constants.pro script is already loaded every time you start up IDL, we can use the constants defined in there. For example, if we wanted to print the mass of the sun in CGS units or Newton's gravitational constant, we'd simply type:
   ```
   idl> print, m_sol
   idl> print, g_n
   ```
5. Those can then, for example, be combined to calculate the freefall time ($t_{\mathrm{ff}}$) of a typical star-forming cloud with a density of $\rho = 4 \times 10^{-19}\,\mathrm{g\,cm^{-3}}$ (which corresponds to a gas number density of about $10^5$ particles per cubic centimetre; how/why?):
   ```
   idl> rho = 4d-19
   idl> t_ff = sqrt ( 3.0*!pi / (32.*g_n*rho) )
   idl> print, t_ff / (1d5*year)
   ```

# 6 Python

Python is now one of the most popular interpreted languages, and the wealth of packages available in Python makes it very well suited for many programming tasks, including scientific computing.

## 6.1 Python startup

1. First, if you do this on an MSO server, we can gain access to python, by adding these lines to `.bashrc`:
   ```
   # --- load modules ---
   module load anaconda texlive
   ```

2. These module-load commands are a very convenient way to load software. Here we are loading a python installation (anaconda) and we are loading latex support (texlive). This will allow access to some of the most commonly used python modules (matplotlib, numpy, etc.), and will allow us to use latex fonts and math typesetting with matplotlib.

3. Now download the python startup package: http://www.mso.anu.edu.au/~chfeder/teaching/astr_4004_8004/material/python_startup.tar.gz
   and copy it to motley into the new `python/` directory.

4. Unpack the tarball. This will create 2 files: `python_startup.py`, and `matplotlibrc` inside a subdirectory. We will go through those files one-by-one.

5. The `python_startup.py` file is run every time you start python, so you can set up some general tasks and important libraries that you want every time you run python.

6. In order for python to know where to look for `python_startup.py`, add these lines to your `.bashrc`:
   ```
   # --- set python environment ---
   export MYPYHOME=$HOME/python
   export PYTHONPATH=$MYPYHOME
   export PYTHONSTARTUP=$MYPYHOME/python_startup.py
   export MATPLOTLIBRC=$MYPYHOME/matplotlib/matplotlibrc
   export MPLCONFIGDIR=$MYPYHOME/matplotlib
   export PATH=$PATH:$PYTHONPATH
   export PATH=$PATH:$HOME/.local/bin/
   ```

7. These settings further attach your python directory in your home dir to the PATH, so you can access common python scripts from anywhere on the filesystem.

8. Finally, I wrote a package to do some common (and more advanced) tasks in python, which is called `cfpack`. You can install this package via `pip install --user --upgrade cfpack` from `PyPI` (https://pypi.org/project/cfpack).

## 6.2 Simple tasks in python

1. Try and do the same tasks as listed in §5.2 for IDL, but now in python. Make use of what's defined in the `cfpack.constants`. You can add to this file later, if you regularly require certain constants or conversions, so you can make calculations very quick and easy.

2. Clarify anything you do not understand about the python startup and basic structure (for example, defs, scope, decorators, etc...).

3. This is a good opportunity to create a git version-controlled copy of your python directory. Please go to §7 for more info.

## 6.3 Creating python modules

1. Start a new python module `mytools.py`. This is where we can add useful functions that we use regularly.

## 6.4 Reading data in python and making map plots

1. Download 3 HDF5 data files (containing column density maps of the world's largest supersonic turbulence simulation):
   EXTREME_proj_xy_000100
   EXTREME_proj_xy_000200
   EXTREME_proj_xy_000300
   These data are column density (2D) projections of the 3D gas density in the simulations. The 3 files are at 3 different times of the evolution. If you are interested in more details of these data and the simulations, please have a look at Federrath et al. (2021) and references therein.

2. Note that HDF5 is version 5 of the Hierarchical Data Format, which is a powerful data format for scientific data. Many codes use HDF5 as the standard now (it also allows parallel I/O). If you have HDF5 installed on your computer (e.g., for Mac users via `sudo port install hdf5`), you can check the content of HDF5 data files simply by typing `> h5ls [file]`. This and `> h5dump -H [file]` prints useful information about the content and header information. Similar to FITS files, HDF5 is a complex file format that can contain multiple datasets of different type, all with header information.

3. First write a new function in `mytools.py` that can be used to read an HDF5 datasets from a file and returns the read data. Make use of the `h5py` library. Call this function `read_hdf5`, and it should take two arguments: the file name and the dataset name. Try calling the function to see if you can read the dataset named 'dens_proj_xy', which contains the column density map.

4. Now write a function that allows you to plot the column density map. Use `matplotlib` and/or `cfpack`. Add a colourbar. Add options in the function to switch the colourbar on/off; and other options to control the minimum and maximum of the data shown, the colour map, and other options you find useful to control when calling the function that plots the map. More advanced: try to produce a logarithmically scaled colourbar.

5. Since we have 3 files, with 3 different column density maps, we can also make use of the `argparse` module to allow us to call the function from the command line and pass the filename as an argument. We can also add options to control the arguments passed to the plotting function. This can all be done with `argparse`.

## 6.5 Re-binning and beam convolution

1. Let's add some beam smearing in order to simulate the effect of looking at these data through a telescope with a finite beam resolution.

2. First we can use the re-binning function provided in `cfpack`. The functions `rebin` and `congrid` (originally defined in IDL), can be used to make re-gridded versions of 2D arrays, for example, to produce lower-resolution versions of the previous images. The `congrid` function works for arbitrary grid interpolations. See if you can understand how the `rebin` function works.

3. However, neither `rebin` and `congrid` simulate Gaussian beam smoothing; they just re-grid a 2D array. Let's implement a Gaussian beam convolution based on the python function `scipy.ndimage.gaussian_filter`. Your own function, which ultimately calls this scipy function for Gaussian smoothing should take the un-convolved map as an input, as well as the Gaussian beam full-width-half-maximum (FWHM) as an input, and return the beam-smeared image. Try with different FWHM values. What is the unit of the FWHM?

4. Show the beam-smeared image(s) as before and compare the non-smoothed with the beam-smoothed images.

5. Add `mytools.py` to your python git repository, if you haven't done that already.

## 6.6 Statistical functions and PDFs

1. Compute the mean, standard deviation, skewness and kurtosis of the log-normalised column density (intensity) of one of the three column-density data from Sec. 6.4,
$$\mathcal{I} = \ln\left(I/\langle I\rangle\right), \tag{1}$$
where $\langle I\rangle$ is the mean column density (intensity) and ln() is the natural logarithm. Use numpy and scipy functions `np.mean`, `np.std`, `scipy.stats.skew`, `scipy.stats.kurtosis`. Explain the relation of mean, standard deviation, skewness and kurtosis to the output of the moments of orders 1–4 computed by `scipy.stats.moment`.

2. Write a function that computes the PDF of $\mathcal{I}$ by calling the numpy function `np.histogram`. I suggest to add this function to `mytools.py`. You will need to adjust the bin locations before plotting, because the numpy function returns the bin edges instead of the bin centres. Note that a PDF is normalised. How exactly?

3. Plot the PDF using `pyplot.hist`; similar can be achieved with `pyplot.bar`.

4. Write a procedure to compute the mean, standard deviation, skewness, and kurtosis of the PDFs by summation over the PDF. Compare the statistical moments of the PDF based on the data and intrinsic functions with those based on the summation over the PDF. Is there a difference? Why? How could this be fixed?

## 6.7 Averaging data, making plots with error bars, and fitting

1. Average all the three column density PDFs over time to produce a time-averaged PDF with error bars.

2. Produce a plot of the PDF including error bars.

3. Fit the time-averaged PDF with a Gaussian function (for instance with `scipy.optimize.curvefit` or better by using `lmfit`). Over-plot the fit.

4. Note that `curvefit` and `lmfit` currently only support errors (uncertainties) in the data direction ($y$ axis), but not in the direction of the variable ($x$ axis). How could you create a fit method that takes errors in both axes into account?

## 6.8 Monte Carlo error propagation

Uncertainty analysis of measurements and derived quantities in astrophysics is extremely important, and published measurement results should always have a proper error analysis associated with them. Here we learn how to do Monte Carlo (MC) error propagation, which goes beyond standard analytic error propagation and can handle non-Gaussian distributions for propagating uncertainties.

1. Suppose you have measured two Gaussian random variables,

$$a = 10 \pm 1 \qquad \text{and} \qquad b = 1.0 \pm 0.1, \tag{2}$$

   and you want to calculate the derived quantity

$$c = \frac{a^2}{b^2}. \tag{3}$$

2. First, calculate the uncertainty of $c$ using standard analytic methods of error propagation.
3. Now write a program that does the MC error propagation. First make Gaussian random numbers (module `random.gauss`) and define $a$ and $b$ based on these Gaussian random distributions. Then define $c$ based on $a$ and $b$.
4. Now plot the PDFs of $a$, $b$, and $c$. What is special about the PDF of $c$? Try also a log-y axis version.
5. Compute the mean and standard deviation of $c$ based on the PDF of $c$ and compare to the analytic estimate. Make sure to implement bin-centered binning for the numerical integration of the PDF, in order to recover the mean and standard deviation more accurately than from the staggered bins.
6. Get the mode (most probable value) of $c$ and the 16th and 84th percentile values.
7. Explore what happens when you replace $a$ with $a = 1.0 \pm 0.5$. Think about the interpretation of writing $\pm$ (standard deviation) when quoting errors/uncertainties, relative to the mathematically correct range for values of $c$.
8. Try changing the sample size of the Gaussians and the number of bins used for the PDFs, in order to study numerical convergence of the results.

## 6.9 Fourier analysis

Fourier transformations are a powerful tool to analyse structure in signals. These signals can be of any kind, but the most common scientific applications involve time-domain (1D) or spatial-domain signals in 2D or 3D. Here we will focus first on a simple example of Fourier analysis of a 1D signal and then move to 2D spatial-mode analysis of column-density data.

1. First construct a discrete signal

$$f(x) = \sin(x) + \sin(10x) + 2 \quad \text{for} \quad x = [0, 2\pi], \tag{4}$$

   using `numpy.linspace` with 200 bins.

2. Plot the discretised version of $f(x)$. What are the characteristic modes in this data?

3. Now compute the discrete Fourier transform of the data, $\hat{f}(k)$, where $k = 2\pi/x$ is the wave number (you can use `numpy.fft`).

4. Plot the power in $\hat{f}$. What does this power spectrum tell us? What is the data structure? Have a look at `numpy.fft.fftfreq` for the wavenumber grid.

5. Now filter the high-frequency component with a top-hat filter, compute the inverse Fourier transform of the Fourier-filtered data and plot it.

6. Create a similar test in 2D, using the function

$$f(x,y) = \sin(x) + \sin(10y) + 2 \quad \text{for} \quad x, y = [0, 2\pi], \tag{5}$$

   again with 200 bins in $x$ and $y$. You can use the `cfpack.get_2d_coords` function to get 2D grids for the $x$ and $y$ coordinates.

7. Display the 2D maps (suggest to make use of the earlier function we made to show maps). Apply different filters to the data and investigate the inverse Fourier transform of the filtered data.

8. Apply this to the column-density maps that we have analysed earlier, e.g., EXTREME_proj_xy_000300. See what can happen to the data when you apply different cutoff wave numbers $k = [(2\pi/x)^2 + (2\pi/y)^2]^{1/2}$ for the top-hat filter. Does this remind you of something?

9. In general: what happens to the mean (average) of the data when you apply a Fourier filter?

## 6.10 Gaussian line fitting

Spectroscopic observations provide measurements of emission or absorption lines. For example, molecular clouds are often observed in emission of the CO molecule. The molecule is excited by thermal collisions and goes into a quantum mechanical state of rotation above the ground state. When the molecule relaxes to a lower state (e.g., its normal rotational state given the temperature of the gas), a photon is emitted, which has exactly the energy difference between the exited state and the state into which the molecule relaxed. For CO, this is often the $J = 1 \rightarrow 0$ transition. The same basic principles hold for other spectral observations, such as for example, HI lines, or emission lines from ionised gas (HII regions) such as the H$\alpha$ line; with the difference that the photons are emitted from different quantum mechanical transitions in the atoms. Since the gas generally moves along the line of sight (LOS), the line is shifted with respect to the zero-velocity frame, due to Doppler shift. Thus, we can use line observations to measure the LOS velocity of the gas, as well as its dispersion (e.g., due to turbulence). The lines are further broadened due to thermal broadening.

Here we want to learn how to fit such spectral line profiles, to obtain the position (velocity) and width (dispersion) of the spectral line. Moreover, since there can be multiple line components (from a mix of different gas dynamics occurring along the LOS), we want to be able to fit multiple such lines. Most commonly, the individual line profiles are described by a Gaussian function with mean and standard deviation, as well as a normalisation constant,

$$I(v) = \frac{I_0}{\sqrt{2\pi\sigma_v^2}} \exp\left[-\frac{(v-v_0)^2}{2\sigma_v^2}\right], \tag{6}$$

where $I(v)$ is the intensity of the line as a function of the velocity $v$ (spectral position or velocity channel), $v_0$ is the mean (due to Doppler shifting), $\sigma_v$ is the velocity dispersion (due to thermal or turbulent broadening or both), and $I_0$ is the normalisation constant of the Gaussian line profile.

1. First, let's create a grid of velocities and define a Gaussian profile $I(v)$ with $v_0 = 0$, $\sigma_v = 1$ and $I_0 = 1$. Try different number of bins for the velocity grid; i.e., make the number of bins a parameter of your script/function. Plot the profile.

2. Now fit this profile with the same Gaussian function, i.e., Eq. (6), and compare the input parameters with the resulting fit parameters. The `scipy.optimize.curve_fit` function provides a simple interface for doing least-squares fitting, but I suggest to use the `lmfit` package, which provides the ability to do composite models (combinations of functions, such the sum of Gaussians), and some added functionality and outputs, such as the reduced $\chi^2$, etc.

3. Add Gaussian noise to the line (in a real observation, noise can come from various sources such as the telescope itself, or the atmosphere through which a line is observed, or the electronics). Re-plot, and re-fit the noisy line. Play with the amount of noise and see if the fit still works.

4. Add multiple line profiles together, where each line profile is shifted by some amount (different $v_0$) and may also have different $\sigma_v$ and/or intensity $I_0$. Re-plot this and play with the parameters of the multiple components (suggest to start with 2 components, then add a 3rd, etc). Depending on the number of Gaussians, try to fit the combined line profile and see if you can create a fitting code that automatically detects the number of Gaussian components that were used to create the combined line profile and that performs a combined fit from which the individual components (their $v_0$'s, $\sigma_v$'s, and $I_0$'s) can be recovered. Try the same with noise added to the spectrum.

## 7 Version control

### 7.1 Basics of version control

1. Imagine you work on a code development project or you write a paper and you'd like to keep track of changes and earlier versions of your code/paper. A neat way to achieve this is to use version control software/tools.

2. Popular version control frontends (partially free or commercial, if you want repositories to be private or shared by a large number of developers) are provided by services such as `https://bitbucket.org` or `https://github.com`. These are primarily webpages that allow you to share your code with others, browse the source code and keep track of changes. For bigger projects, you can also establish teams that work on the different pieces/modules of the same code.

3. To get started, you need to create an account with `https://bitbucket.org`. I suggest to use your ANU email address when signing up, which gives you access to unlimited private repository users (otherwise, you'd have to pay for that).

4. A key element of these version control systems is that they keep their own files inside the directory(ies) of your code, in order to store and update changes – basically to keep an entire history of what's been going on with the code; who made changes, what changes, and when. They also allow you to revert to previous versions in case some bugs slipped into the code or something broke at some stage.

5. In order to start a versioned code, you will need to install a version control system or version control software. Some of the first bigger ones were Concurrent Versions System (`cvs`) and Subversion (`svn`). Nowadays Mercurial (`hg`) and Git (`git`) are popular version control systems. Here we will focus on `git` with some examples.

## 7.2 Starting a `git` repository

1. To get started with `git`, you have to install it on your computer. Then we pick a directory with source code or any files that we want to keep under version-control and change into it.

2. This is how we start a Git repository in that directory (for example in `mycode/`):
```
> cd mycode/
> git init .
```

3. Then type `> git add [file1] [file2] [...]` to add all of the relevant files that you want to keep under version control.

4. Now we can check the status of the files by typing `> git status`. This brings up a list of changes and a list of files that are in the directory (and subdirectories), but that are not under version control. You can create a hidden file called `.gitignore` and add all of the files that you want git to ignore, so they don't annoy you every time you type `> git status`.

5. Finally, once you are happy with the changes (say you added all files or if they were added earlier you may have modified them when you develop your code further), you type
```
> git commit -m 'message describing change(s)' [file_to_be_commited].
```

6. This last command commits the change to the file and creates a new version in the system, which you can revert to later. Or when you make further changes to the same file, you can compare it to the previously committed version by typing:
```
> git diff [file_to_check_changes_since_previous_commit]
```

## 7.3 Uploading/Communicating repository to server

1. Up to this point, the version-controlled code just lives on your own computer, but we also want to upload the code to a safe location on a server, in case something happens to your own computer, but also in case we want to share the code with specific people or with the entire public.

2. To do this, we can create an account on `https://bitbucket.org` and start a new repository or import the existing repository from the previous steps. Note that if you sign up with your ANU email address, your bitbucket account will automatically be an academic account, which means that you can add an unlimited number of users to private repositories and won't have to pay for it – otherwise it costs something like USD $2 per user when exceeding 5 users :-( ...so better take advantage of being part of the Uni!

3. Once configured correctly, i.e., providing the correct URL and paths, such that your computer knows that it should upload changes in the local repository to the bitbucket or github server, you can simply type
```
> git push
```
to push all the changes in the local copy to the server. This will then allow you to browse the code online, to share it and to view changes to the version-controlled code in an

internet browser (basically, it's just a nicer view of what you can get with `> git status` and `> git diff` on your local working copy). You can also upload your public key (see §2.6) in your bitbucket account, so you don't have to type your bitbucket password to do commits, pulls, pushes, etc.

4. Having the code on the server will also allow you to share it more easily with others. There are different options to do this, for example, cloning a repository, or forking or branching a repository. We won't go into details of that, but essentially, this allows one to get a copy of the repository and continue working on it within their own local copy, such that the global source repository is not damaged. However, changes by another user that are deemed useful for the global copy of the repository can be merged in by issuing a so-called *pull-request*.
5. In summary, when you develop code, it is a good idea to use a version control system. What seems awkward at first is actually extremely useful once you get into trouble with keeping track of changes based on your own strategies (e.g., keeping many earlier copies of the same file/code). Version control software provides a standardised way of keeping different versions of a code or simply a bunch of files that undergo regular changes.

### 7.4 Overleaf

Overleaf is a useful collaborative latex writing environment. Since 2023, ANU has an enterprise agreement with overleaf, which means that you can use your ANU account (UID, SSO) to access the advanced features of overleaf at no cost (such as track changes). More info here: `https://www.overleaf.com/edu/anu`.

Overleaf supports `git`. In case, you want to write offline and with your own latex environment, you can do so and then upload/sync to overleaf. You can also edit on overleaf and then git pull to your local working copy. This is useful for backup and offline work on latex documents managed by overleaf.

## 8 Automation with python

### 8.1 Automation of web page interactions

Automation of browser/web actions can be achieved via the python module `selenium`, using a so-called `webdriver`. Basic functionalities are available in `cfpack.browser_control` using the Google Chrome driver.

### 8.2 Automation of operating system functions

One can more generally automate any mouse or keyboard interaction in Mac OS or Windows (not sure about Linux, but probably yes) with the `pyautogui` module.

- First need to install `pyautogui` and `opencv-python`.
- `import pyautogui as pag`
- pag.FAILSAFE = True # move to any screen corner to end automation
- pag.PAUSE = 0.1 # wait time between pag calls
- pag.size() # get screen width and height
- pag.position() # current position of the mouse