ASTR4004/ASTR8004
Astronomical Computing
Lecture 06

Christoph Federrath

12 August 2019

Extracting Data from Plots, Version control, Interactive Data Language (IDL)

1 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.

2 Version control

2.1 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 `http://www.bitbucket.org` or `http://www.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. 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.

4. 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.

2.2 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]`

2.3 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 `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 lecture 03) in your bitbucket account, so you don't have to type your bitbucket password to do commits, pulls, pushs, etc.

4. However, having the code on the server will also allow you to share it 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.

3 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.

3.1 Getting started

1. Login to `malice` 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 malice 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 everytime you start IDL.

3.2 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 be combined to say calculate the freefall time ($t_{\rm ff}$) of a typical star-forming cloud with a density of $\rho = 4 \times 10^{-19}\,{\rm g\,cm^{-3}}$ (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)
   ```