The NEMO package (example report)

[1] The NEMO package is a software toolbox written primarely in C, with some code in C++ and Fortran, and thus not suitable for PHYS265. It provides a library, with which new programs can be written. The philosophy of this is to emulate the Unix environment, whereby each program provides a unique function, and using Unix pipes a filter can be created to solve astrophysical problems, in particular the N-body problem in the case of NEMO. Other tools were developed to work on orbits, images and tables, making it a very versatile environment to solve a variety of astrophysical simulations.

[2] I picked this package because I was one of the co-authors and wanted to provide an example of such a report for the 265 final project. The 265 projects have to be python based, and although NEMO does have example code in python, it does not rely on it.

[3,4] NEMO was written in 1986, by Barnes, Hut and Teuben at the Institute for Advanced Study in Princeton. It has been maintained by Teuben ever since, although Barnes developed NEMO into ZENO and still maintains that. Hut wanted to solve a different type of N-body problem and wrote the Starlab package with McMillan, Makino and others. The Starlab package then evolved into the AMUSE package, which is a very versatile python package that could be used as a challenging PHYS265 project!

There are actually many N-body codes that have been written, a large number can be found on ASCL (both using the term nbody and n-body), a number of them are available through NEMO, so interesting comparisons can be made.

[5] Although installing these types of packages can be notoriously hard on Unix, NEMO was relatively easy to install because the libraries needed (mostly the PGPLOT plotting library) are generally available on Linux and Mac. The authors have also gone through great length to use the autoconf library to simplify the install. The install took about two minutes.

```
git clone https://github.com/teuben/nemo
cd nemo
./configure
make build check bench5
source nemo_start.sh
```

after this NEMO commands such as `mkplummer` can be used. The accompanying notebook shows this is more detail.

[6] There is a small python component in NEMO, which can be installed with the typical pip install command

```
 cd $NEMO
 pip install -e .
```

[7] Source code is available on github (see item 50. There are currently no binaries available. Making for example a docker image available could help lowering the barrier to use it. After the package was installed, a runtime check and benchmark were suggested to be used to confirm the package was correctly installed, and provide an idea how fast the computer is.

[8] Parts of NEMO are also used by galpy, notably the flexible way to describe potentials. The glnemo2 visualization software and the unsio library also use the NEMO file format to describe visualize an N-body system.

[9] In my example notebook I use the `bash` scripting language in a Jupterlab Desktop (JDL) bash, but here is a one line example how to create a Plummer sphere with 10 particles and print the positions and velocities:

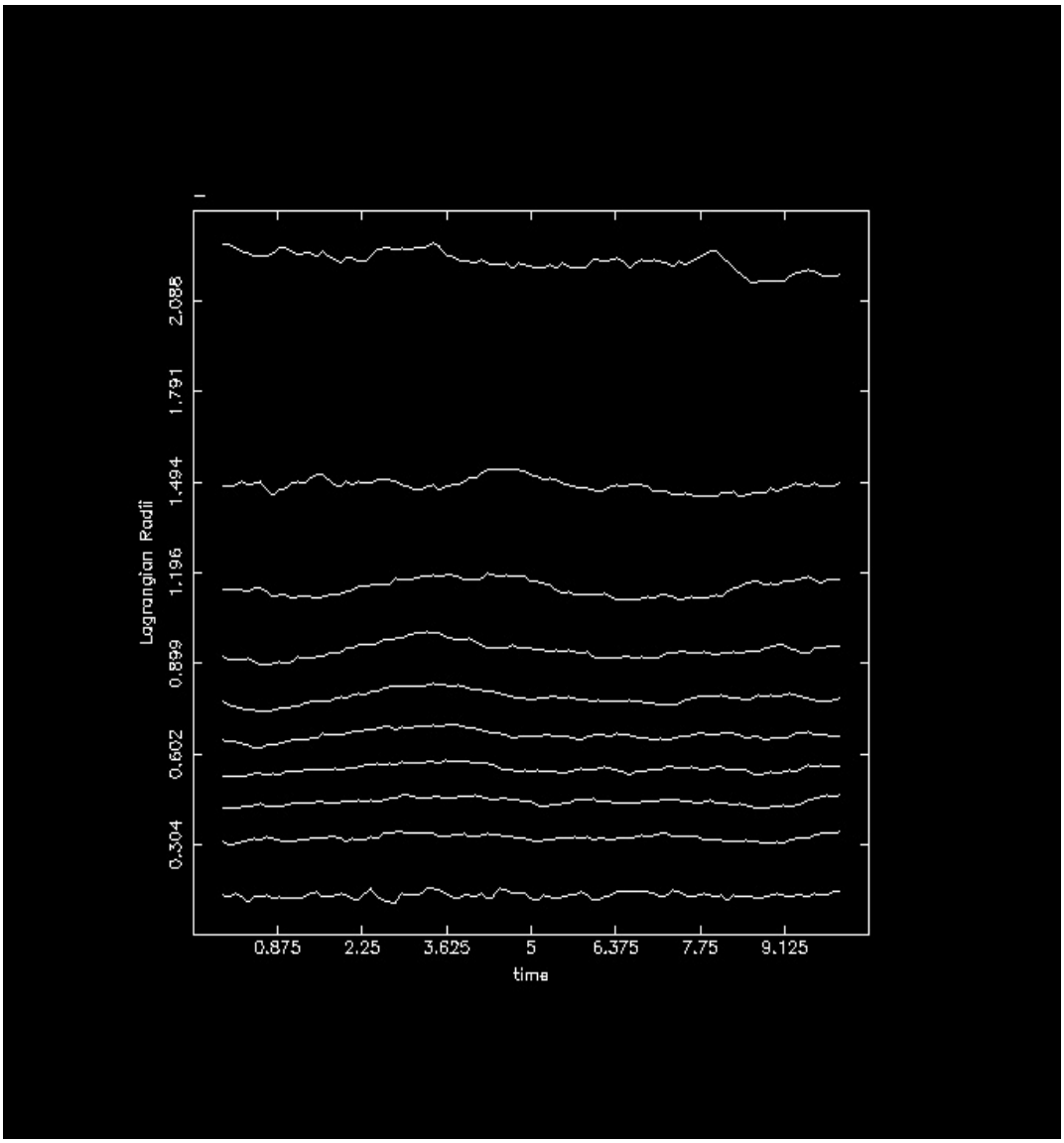
```
 mkplummer - 10 | snapprint -
```

also showing how Unix pipes can be use in NEMO.

[10] The accompanying notebook shows how I have used the classic Aarseth code, but integrated into NEMO. It is a `bash` notebook, not `python` for obvious reasons.

[11] NEMO programs use the `yapp` API to produce figures. At installation time the user needs to select a working yapp interface. I selected the pgplot interface, because it can at run time select an X11, or PNG or PS output device. For the generic `tabplot` table plotter program, there was also a simple matplotlib based tabplot.py available, which is OK for some interactive plotting, but is not mature yet.

[12] here is a figure of the Lagrangian radii showing 1,10,20,...90,99% of the total mass as function of time in a 2048 Plummer simulation, made with nbody1.



[13] There is still very little python in NEMO, most work is done via unix style commands, and users writing shell scripts for their workflow.

[14] The input for NEMO can be an image, an N-body snapshot or most ascii table formats. Some programs exist to create such data from scratch, e.g. `ccdgen`, `mkplummer`, or `tabgen`.

[15] The output is likewise. image, snapshots, tables and orbits can be produced.

[16,17] After the code was installed, it was suggested to run the commands `make check bench5` which run a standard set of tests and benchmarks to make the user more confident the code is working and producing the correct answers. The contents of the data files are compared (on a statistical level) with a baseline version using the NEMO `bsf` program. Relying on the benchmark here to ensure it is working.

[18] NEMO does use matplotlib and numpy, but the main package depends on a plotting library (often pgplot) and optionally on a number of scientific libraries such as hdf4, hdf5, cfitsio, netcdf, gsl.

[19] The documentation was extensive. NEMO still uses the Unix man format, accessible via the `man` command, but these manual pages are also available in html, and include hyperlinks on https://teuben.github.io/nemo/man_html/index1.html. An original latex based manual was converted to `readthedocs` (built by python) and probably provides the best introduction to the package. https://astronemo.readthedocs.io/en/latest/

[20,21] References

- NEMO preferred citation: https://ui.adsabs.harvard.edu/abs/1995ASPC...77..398T
- zeno: https://ascl.net/zeno
- starlab: https://ascl.net/starlab
- AMUSE: https://ascl.net/amuse
- glnemo2: https://projets.lam.fr/projects/glnemo2/wiki
- unsio: https://projets.lam.fr/projects/unsio/wiki/Wiki
- pgplot: https://ascl.net/pgplot
- galpy: https://ascl.net/galpy

[22] Using ADS only two were identified in the ASCL reference: https://ui.adsabs.harvard.edu/abs/2010ascl.soft10051B/citations But on the actual originating 1995 paper there are 179 citations: https://ui.adsabs.harvard.edu/abs/1995ASPC...77..398T/citations

This shows one of the complaints of software being in two references on ADS.

[23] Not applicable to me, so the answer is no. there was nothing new here.

Original checklist of items to weave into your report

1. name of the package, and what is the basic aim of what the package does or solve?
2. why/how did you select this package?
3. how old is the package? does it have a geneology, i.e. what related codes came before or after. are there other codes that solve the same problem?
4. is it still maintained, and by the original author(s)?
5. evaluate how easy it was to install and use. What commands did you use to install?
6. does it install via the “standard” pip/conda, or is it more complex?
7. is the source code available? For example, “pip install galpy” may get it to you, but where can you inspect the code?
8. is the code used by other packages (if so, give one or two examples)
9. give an example how you use the code. Is it commandline, or jupyter notebook, or a web interface?
10. provide examples using the code. if you prefer to use a jupyter notebook instead of a python script, that’s ok.
11. does the package produce figures, or are you on your own? Is matplotlib used?
12. your code and report should show at least one figure, and create a nice figure caption explaining what it shows. You notebook should show how the figure was made (reproducable)
13. is the package pure python? or does it need accompanying C/C++/Fortran code?
14. what is the input to the package? Just parameters, or dataset(s), or can they be generated from scratch?
15. what is the output of the package? Just parameters, or dataset(s)?
16. does the code provide any unit tests, regression or benchmarking?
17. how can you feel confident the code produce a reliable result? (see also previous question)
18. what (main) python package(s) does it use or depend on (e.g. numpy, curve_fit, solve_ivp) - how did you find this out?
19. what kind of documentation does the package provide? was it sufficient for you?
20. if you use this code in a paper, do they give a preferred citation method?
21. provide any other references you used in your report.
22. can you find two other papers that used this package. E.g. use ADS citations for ASCL based code.
23. did you have to learn new python methods to use this package?