ASTR4004/ASTR8004
Astronomical Computing
Assignment 4 (exam assignment)

Christoph Federrath, Yuan-Sen Ting, Michael Ireland

due Thursday, 9 November 2023, 23:59pm

Important Information

This is the exam assignment. It is due on Thursday, 9 November 2023, 23:59pm. Please submit your solutions via Turnitin. Produce all necessary submission files for each section, name them to clearly indicate the section (question) number, and make a tarball named <Uni-ID>.tar.gz, containing all submission files. Upload the single final tarball to Turnitin. Since this is the exam assignment, we cannot accept late submissions! Please make sure to submit on time.
As you can see below, there are 6 questions in total. Each of the questions is worth 10 points. You may choose any 4 of the 6 questions. Thus, the maximum possible total you can get is 40 points, which would be 100% for the exam assignment. You may also choose to submit solutions to more than 4 questions, however, only your best 4 solutions will count towards your total score, i.e., those 4 solutions with the highest points in total (in other words, you cannot have points from any more than 4 questions contribute to your final total).

1 Question 1

Please refer to the associated tarball for exam question 1 (directory q123).

(10 points)

2 Question 2

Please refer to the associated tarball for exam question 2 (directory q123).

(10 points)

3 Question 3

Please refer to the associated tarball for exam question 3 (directory q123).

(10 points)

4 Image processing and Fourier transformation

Here you will make a python program that reads a column density map of a molecular cloud called 'The Brick' near the Galactic Centre (you can read more about this cloud in Federrath et al., 2016), apply mirroring and zero-padding to the image, compute the Fast Fourier Transform (FFT) with numpy or scipy, make a Fourier image and compute the power spectrum of the column density map.

1. Download the observational column density map from `http://www.mso.anu.edu.au/~chfeder/teaching/astr_4004_8004/material/brick.fits`. Use the astropy library to read the data map in the fits file (`http://docs.astropy.org/en/stable/io/fits/`) into a numpy array.

2. Make a python function to produce an image of the map with a colour bar and write the image to a pdf file named 'brick.pdf'. See the left-hand panel of Figure 1 for an example thumbnail image of how this should look like.

3. Use the numpy functions `np.fliplr` and `np.flipud` to produce a mirrored array and image. Write the image to a pdf file called 'brick_mirrored.pdf' (see the middle panel of Figure 1 for a thumbnail).

4. Now use the numpy function `np.pad` to pad zeros symmetrically to the left and right of the image, such that the total dimensions become (1278, 1278). Make an image of this called 'brick_mirrored_zped.pdf' (see Fig. 1 for how this should look.)

5. Make a 2D FFT of the mirrored-and-zero-padded column density map. Shift the $\mathbf{k} = (0, 0)$ position to the centre of the Fourier image and write out an image called 'brick_fourier_image.pdf'. The result of this should look like the last panel of Figure 1.

6. Compute the 1D power spectrum $P(k)$ of the mirrored and zero-padded column density, where $k = \sqrt{k_x^2 + k_y^2}$. Make a log-log plot of the power spectrum, $P(k)$, and write this out as an image called 'brick_power_spectrum.pdf'.

Put everything into a single Bash-shell-executable python script that runs the entire analysis with the input file (the column-density fits file) sitting in the same folder.

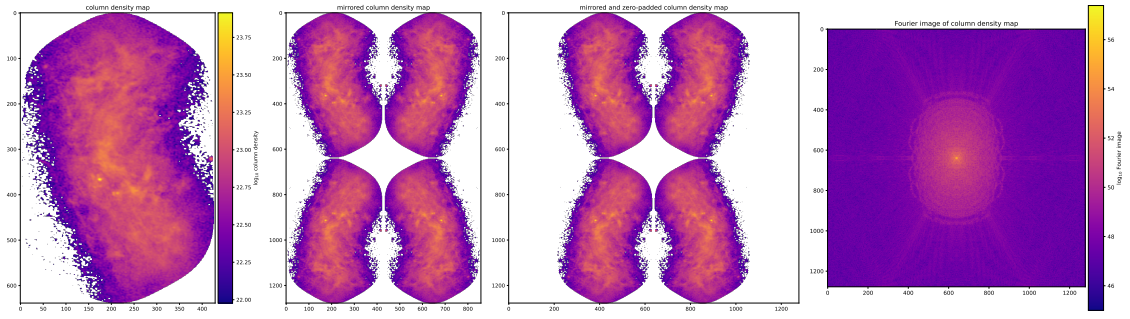

Figure 1: Left to right: original column density map, mirrored, zero-padded, and $\log_{10}$ Fourier image. Make sure to reproduce these not so small as in this assignment, but with readable font sizes; these are just meant as thumbnails to give you some idea of what the output of your script should look like.

The script should automatically produce the images with the requested filenames above (original column density image, mirrored, zero-padded, and Fourier image), as well as the final plot of the 1D column-density power spectrum.

(10 points)

5 Parallel computing and scaling

Here we write an MPI-parallelised python program (using mpi4py), test its parallel scaling performance, and determine its parallel fraction.

1. Start by setting up MPI: get total number of ranks, each rank's ID, etc.

2. Let the Master rank (only the master rank!) read the numpy array in file 'q5.npy' (see associated tarball), using np.load(...). Check its shape.

3. Use cfpack.congrid to interpolate the data to increase the size of the array by a factor of 4 in each direction. You should end up with a shape of (5112, 5112). Pretty big!

4. Use comm.Send and comm.Recv to communicate the shape of the array (which only the Master rank knows at this point) to all ranks. Hint: the program must work for any number of MPI ranks, so use a loop over all ranks on the Master rank to send the shape to all other ranks. All ranks (except the Master) need to receive.

5. Now plan to decompose the total size of the data into equal parts (as best as possible; take care of the fact that depending on the number of processors, a perfectly equal division will not always be possible).

6. Let the Master rank send the respective decomposed parts of the data to the respective rank that needs to receive that part. Use again comm.Send and comm.Recv to achieve this.

7. Now let each rank sum over their respective part of the data (using a loop over all the datapoints local to the rank), such that you can compute the total mean and the total standard deviation of the data, in a parallelised way. You will need to use comm.Allreduce after the local looping, in the process.

8. Let the Master rank print the mean and standard deviation computed in this parallelised fashion, to the screen. Compare with the mean and standard deviation of the original (full, non-decomposed) data (the latter must always remain on the Master rank only!).

9. What type of parallelisation paradigm did we use here (SISD, MIMD, SIMD, or MISD)?

10. The entire script needs to be timed, so you can record the run time, from MPI setup, to printing out the final mean and standard deviation. This will allow you to measure the runtime when testing the code with different numbers of total MPI ranks.

11. Now run your code on motley.anu.edu.au, using mpirun. Run it on 1, 2, 4, 8, 16, and 32 cores on motley, and record the runtime for each of those cases.

12. Make a plot of speed-up factor (relative to the runtime of a single rank) versus number of ranks. The x-axis should be in log scaling.

13. Now fit Amdahl's law to the data and determine the parallel fraction of your code.

14. Over-plot the best-fitting Amdahl's law.

Provide all necessary scripts/code and produce a write-up including the final figure.

(10 points)

6 Markov Chain Monte Carlo

In this assignment you will use `emcee` (https://emcee.readthedocs.io, or a similar Monte Carlo Markov Chain package) in python. You will simulate a data set with a periodic component and fit a function to it. This could be, for example, a photometric dataset from Kepler, or a series of radial velocity points. Some skeleton code (with many gaps!) is available here: http://www.mso.anu.edu.au/~chfeder/teaching/astr_4004_8004/material/mcmc_assignment_hints.py.

1. Create a function using python and `numpy` that simulates data originating wth a physical model:

$$v = a_0 + a_1 t + a_2 \sin(a_3 t + a_4), \tag{1}$$

   with additive Gaussian noise. You should simulate data at a uniformly distributed set of times over an interval $[t_1, t_2]$. The functions should include the inputs a_i in the form of a 1-dimensional python numpy array, the time interval, the number of points and the standard deviation of the additive Gaussian noise.

2. Setting $a_0 = 0$, $a_1 = 0.5$, $a_2 = 1$, $a_3 = 1$ and $a_4 = 0$, simulate a data set with times uniformly distributed from $t = 10$ to $t = 30$, containing 100 points with Gaussian errors with standard deviation 0.4. Plot these simulated data, together with the original noiseless physical model computed over the interval.

3. Use `emcee` or a similar package to fit to this simulated dataset. Plot histograms of the fitted parameters – do the results make sense? Are any of the parameter fits correlated? Try this again for a simulated dataset with $a_4 = 3$.

4. Show that the following is a re-parameterisation[1] of Equation (2):

$$v = b_0 + b_1(t - 20) + b_2 \sin(b_3 t) + b_4 \cos(b_3 t) \tag{2}$$

[1] Re-parameterisation means that the b_0 through b_4 can be written in terms of a_0 through a_4.

Why is Equation (2) better than Equation (1) for a reliable Monte-Carlo Markov chain computation? The second set of parameters above ($a_0 = 0$, $a_1 = 0.5$, $a_2 = 1$, $a_3 = 1$ and $a_4 = 3$) illustrates the difference well. Does it reduce any of the parameter correlations, and if so, why?

5. Your colleague has a physical model that is Equation (1), with uniform priors in all parameters. However, they use Equation (2) in a Monte-Carlo Markov chain run instead with uniform priors, and then compute the distribution of the a_i parameters from the chains of the b_i parameters. This produces an implicit prior on a_2, biasing the posterior and giving an incorrect results. What is this implicit prior?

Include all python code in your assignment, as well as a write-up.

(10 points)

Maximum Total: 40 points (sum of best 4 solutions)

References

Federrath, C., Rathborne, J. M., Longmore, S. N., et al. 2016, *Astrophys. J.*, 832, 143