Assignment 1, Deadline 25 October 2019

General Remarks

- The complete assignment must be written as a single Jupyter notebook that contains the code, relevant output, graphcis and all explanations.
- The Notebook must be executable without errors. To make sure that this is the case, before submitting reload the notebook and run it from scratch to make sure it runs fine. Any submitted notebook that does not run correctly will receive an automatic 0 mark.
- While you can (and should) include in your notebooks results from longer runtime, make sure that the parameter choices in your notebook on submission are chosen such that the notebook does not take longer than 2 minutes to run on your computer. We are most likely using much faster machines for marking. But notebooks that have a substantially longer running time will be rejected.
- You must follow in your code PEP8 coding guidelines with the only exception that lines can be longer than 80 characters. But please do not make them much longer than 100 characters so that they are still readable (consider at a soft limit). Failure to adhere to PEP8 coding guidelines can result in substantial point reductions in severe cases.
- Correctness of the codes will count 40% of the mark. You will receive 30% for presentation, which includes sensible graphs and thorough documentation. The remaining 30% will be for efficient coding. However, a submission without documentation will still count as a fail as this is a mandatory component.
- You will be able to submit your Jupyter notebooks from a few days before the deadline onwards to the Moodle course page. A corresponding announcement will be made in time.

The following two questions are computational exercises that ask you to try to write as efficient implementations as possible. Tools you are allowed to use are Python with Numpy, Scipy, Numba, Numexpr, Multiprocessing and Matplotlib for output. For both exercises describe how you optimise the code and the performance benefits you have achieved. It is easiest to start with a simple Python implementation and then to start optimising it. Describe the reasons and benefits for the various optimisations that you have chosen.

Question 1: Random walk on a lattice
Consider a lattice on the unit square with N points in each coordinate direction (including points on the boundary of the square). We place a particle on a starting position $(x_i, y_j), i = 1, \ldots, N; j = 1, \ldots, N$ on the lattice and start a random walk from there. In each step of the random walk the particle can go with equal probability one step left, right, up, or down on the lattice. We stop the random walk when one of the boundary points is reached. For each interior lattice point we now want to compute the probability that the particle reaches the bottom edge of the unit square. The expected output is a plot of the probability function $p(x_i, y_j)$ over the lattice. To estimate the probability for each lattice point you need to run a number of random walk instances from that point and record the fraction of random walks that end up at the bottom edge. Think carefully about how you optimise the code. Study the convergence of your code as you increase the number of random walk trials per lattice point, and also as you increase the number N of lattice points. In your documentation justify your implementation decisions.

Question 2: Potentials generated by random particles.
We consider the unit circle in two dimensions with radius 1 centered at the origin. We now place a number N of potential unit charges at random positions y_j inside the unit circle. Each potential generates the field $u_j(x) = \frac{1}{2\pi} \ln |x - y_j|$. for $x \neq y_j$. Hence, the total field at a given point x outside the unit circle as computed as $u(x) = \frac{1}{2\pi} \sum_j \ln |x - y_j|$. We are interested in the function

$$f(r) = \frac{1}{2\pi r} \frac{1}{N} \int_{C_r} u(s) ds,$$

which is the average value of the combined potential u over a circle with radius r. To approximate the value $f(r)$ we can use a trapezoid rule to obtain

$$f(r) \approx \frac{1}{NM} \sum_{j=0}^{M-1} u(x_\tau)$$

with the x_τ being M points equally spaced in angle on the circle with radius r. Implement an efficient method to compute and visualize $f(r)$ for $1 < r < 10$.

A single particle at the center of the unit circle produces the potential $u_0(r) = \frac{1}{2\pi} \ln r$ for $r > 0$, uniform in each direction.

By performing trials with random placements for the positions y_j in the unit circle estimate the expected value $e(r) = E[\frac{|f(r) - u_0(r)|}{u_0(r)}]$ for the relative difference of $u_0(r)$ and f for $1 < r < 10$. Again, carefully describe your code implementation strategy, optimisations, and the influence of the parameters N, M and the number of random trials.