# Computational Physics: PHAS0030, 2018-2019
# Problem Sheet 4

*Hand in your answers by* **Monday 22 April**, *on Moodle through the link provided. Marks per section are shown in square brackets. Your answers should ideally be a Jupyter Notebook, though Python code is acceptable. Ensure that you comment your code, and give every function a docstring.*

1. You will be performing a molecular dynamics simulation of the Lennard-Jones model of Ne, as we saw in lectures.

   (a) Based on your work in lectures, set up a lattice of 64 Ne atoms on cubic grid points, using a box length of $1.1\times 2^{1/6}\sigma$ (slightly larger than we used in class). Make sure that you also have defined an appropriate function to calculate the energy and forces for the Lennard-Jones potential (you may take this from the classwork). [2]

   (b) Initialise the velocities so that they give the system an initial temperature of 50K, using the fact that the average kinetic energy per particle is $3k_BT/2$. [2]

   (c) Create an array to store the mean squared displacement (a single number) at each step of the simulation. Now run the simulation for 10,000 steps with a timestep of $10^{-15}$s (this make take a few minutes!), storing the displacement and velocity at least at each step. Ensure that you apply periodic boundary conditions (as we did in class). (The mean squared displacement is defined as:

   $$D(t) = \frac{1}{N}\sum_{i=1}^{N}\left(\mathbf{r}_i(t) - \mathbf{r}_i(0)\right)^2 \tag{1}$$

   which can be calculated efficiently with `np.sum` and simple numpy array arithmetic.) Plot the displacement, and estimate the gradient of a straight line fit to the data (do this by eye, to one significant figure). [4]

   (d) Now calculate the following integral:

   $$I(t) = \frac{1}{3N}\int_0^t \sum_{i=1}^{N} \mathbf{v}_i(t)\cdot\mathbf{v}_i(0)dt \tag{2}$$

   with a simple numerical rectangle integration. Plot this quantity, which should converge with time. Estimate the value, and see how well it compares to $\frac{1}{6}$ of the value from the previous question (when fully converged, the two should agree). [2]

2. You will model a ferromagnet using Monte Carlo, as we did in lectures, but based on an alternative update procedure where two spins selected at random are swapped.

   (a) Set up a random spin array as we did in lectures with a box length of 50, and plot it to check that it looks reasonable. [2]

   (b) Write a function to perform a trial update on the spin array, taking two locations as parameters. Calculate the energy change when the two spins are swapped (you will need to work this out carefully on paper first, and document it in your notebook) and accept it according to the Metropolis algorithm that we discussed in lectures. [2]

   (c) Now perform a 50,000 step Monte Carlo simulation. At each step, choose two lattice points at random and call your update function. Keep a record of the overall total energy. Every 5,000 steps (or more often if you want) plot the sample, using subplots (`fig = plt.figure(); ax = fig.add_subplot(3,4,index); ax.imshow()` or similar). [4]

   (d) Plot the total energy against timestep, and comment *briefly* on how this compares to the spin-flipping approach we used in the further work of Session 8, Section 6 (for instance, which is more efficient?). [2]

3. You will apply fast Fourier transforms (FFTs) to the output from the previous question. (If you could not find a result from the previous question, then copy and paste the model solution from the Session 8 Further Work to generate the spin array.)

   (a) Using `np.fft.fftn`, perform a Fourier transform of the spins in 2D. Plot the coefficients (note that you can shift the output array from the FFT so that the zero wavelength is in the centre using the command `np.roll(np.abs(tmp),(25,25),(0,1))`, which assumes an array size of 50); plot the coefficients for the *starting* array of random spins in a second image. Comment briefly on the differences between the two, thinking about where the largest Fourier components are located, and how this compares to the structures of the spins. [3]

(b) Now calculate the k-vectors (use lines like `k2darr[0:nk2dmax] = dk2d*np.arange(0,boxlen/2)` and `k2darr[nk2dmax:] = dk2d*np.arange(-boxlen/2,0)` after defining `dk2d`; then use `np.meshgrid` to generate an appropriate set of 2D vectors). Use these vectors to calculate the $x$ and $y$ derivatives of the array (scale the FFT by $ik_x$ or $ik_y$ and apply the inverse FFT). Create an array $\sqrt{(\partial f/\partial x)^2 + (\partial f/\partial y)^2}$ and plot it. Taking a derivative like this is one approach to detecting edges in images; comment briefly on how effective you think that it is. [3]

(c) We can take a second derivative by scaling the Fourier transform by $k^2$ and reversing the transform. Do this and plot the result. [2]

(d) Define an array, say `a`, in real space (the same size as your spin array) with zeros everywhere except: `a[0,0] = -1` and `a[0,1], a[0,-1], a[1,0], a[-1,0]` all set to 0.25 (you may recognise this as the finite difference 'stencil' for the second derivative). Take the Fourier transform of this array and multiply by the Fourier transform of the spins. Perform the inverse Fourier transform and plot. Comment on how this relates to your answer from the previous question. [2]