PHYS358: Homework 7 (due Nov 20, 9:30am)

PDEs: Poisson Solvers: Fourier Method

Note: Please read the complete homework instructions before you start.

The Convolution View:

Despite the power of iterative schemes (Jacobi, Gauss-Seidel), and specifically of multigrid solvers, there are cases where we can deal with the problem of solving elliptic PDEs much more efficiently. One such technique makes use of the *convolution theorem*, or the fact that for two (periodic) functions $f(x), g(x)$, their convolution can be calculated via the product of their Fourier transforms (see Numerical Recipes for more details).

In the following, we'll step through some of the math behind this argument, and we will develop a simple implementation of a Fourier solver.

(7a) The Convolution Theorem [10pts]: The Poisson equation can be physically motivated by calculating the potential generated by a charge (or mass) distribution via integrating over that mass distribution,

$$\Phi(\boldsymbol{x}) \equiv \int \frac{\rho(\boldsymbol{x'})}{|\boldsymbol{x'}-\boldsymbol{x}|} d^3x' \tag{1}$$

$$= \int K(\boldsymbol{x},\boldsymbol{x'})\rho(\boldsymbol{x'})d^3x', \tag{2}$$

i.e. the mass density $\rho(\boldsymbol{x})$ is *convolved* with a (symmetric) kernel function $K(\boldsymbol{x},\boldsymbol{x'})$. You probably have met K elsewhere under the name Green's function. The problem with eq. 1 is the same as with the corresponding force equation: if we have N particles to sum over to determine the force (potential), we need $\mathcal{O}(N^2)$ operations (remember the Kepler problem in our discussion of ODEs). Therefore, the goal eventually is to find a more efficient algorithm.

To start out, prove the convolution theorem, i.e. show that for two functions $f(x), g(x)$,

$$\mathcal{F}[f]\,\mathcal{F}[g] = \mathcal{F}[f * g], \tag{3}$$

with the Fourier transform

$$\mathcal{F}[f](k) \equiv \int_{-\infty}^{\infty} f(x)e^{-ikx}dx, \tag{4}$$

and the convolution

$$[f * g](s) \equiv \int_{-\infty}^{\infty} f(x)g(s-x)dx. \tag{5}$$

(7b) Reviewing the discretizing of the Fourier transform: DFT [5pts]: As always, we need a discretized version of our problem – in this case eq. 4. Show that the discrete Fourier transform of a function $f(x)$ periodic over the interval $x = [0, L]$ can be written as

$$F_k = \sum_{j=0}^{J-1} f_j\, e^{-i2\pi jk/J}, \tag{6}$$

with $f_j \equiv f(x_j)$ and $x_j = jL/J$. Start with eq. 4 and discretize.

(7c) The normalization of the reverse Fourier transform [5pts]: If F_k is the discrete Fourier transform of f_j, show that the inverse Fourier transform applied to F_k results in

$$f_l = J\mathcal{F}^{-1}[\mathcal{F}[f_j]], \tag{7}$$

i.e. when inverting F_k, you gain a factor J (number of elements). *It may be easiest to write this down for $J = 4$, $l = 0$. Remember that $e^{i2n\pi} = 1, e^{i(2n+1)\pi} = -1$, for $n = 0, 1, 2, ...$*

(7d) Implementation of a "Slow" Discrete Fourier transform [15pts]: The discrete Fourier transform (eq. 6) can be written as

$$\boldsymbol{F} = \boldsymbol{W}\boldsymbol{f}, \tag{8}$$

where $\boldsymbol{W}$ is a (n, n)-matrix with the components $W^{kj} = e^{-i2\pi jk/J}$, where $W \equiv e^{-i2\pi/J}$. Implement this transformation in the routine `sft` taking a vector of length J and returning its transform. `sft` should also take an argument specifying the direction of the transform (forward or backward). Make sure your inverse transform is appropriately normalized. *Hint: Eq. 8 results in an array $\boldsymbol{F}$ of complex numbers. To plot the amplitudes of $\boldsymbol{F}$, use* `np.abs()`.

(7e) Tests [10pts]:

1. Calculate (analytically!) the Fourier Transform of

$$f(x) = \sin(2\pi nx), \tag{9}$$

   with n a natural number.

2. Test your routine `sft` with the sine as above. Your code should display the amplitude F_k as a function of the wave number k, and it should over plot the original function, and the inverse of its Fourier transform (obviously, the two should agree). I recommend $J = 32$. Try $n = 1, 4, 8, 16, 17, 24, 32, 33$. Discuss the results.

3. How does the computing time requirement of `sft` scale with the number of points J?

The Finite Difference View: While it is possible to solve the Poisson equation via convolution, this usually means calculating the FT of the Greens Function, which can get tricky when considering face vs cell centered quantities. Another way (and actually a way more elegant way) is to write out the finite difference version of the Poisson equation and transform that expression into Fourier space.

Just to remind ourselves: We are given a density distribution $\rho(x)$ and would like to calculate the potential $\Phi(x)$ from

$$\nabla^2\Phi(x) = 4\pi G\rho(x). \tag{10}$$

The finite difference expression for our 1D system reads

$$\Phi_{j-1} - 2\Phi_j + \Phi_{j+1} = 4\pi G\rho_j(\Delta x)^2. \tag{11}$$

(7f) The 1D Kernel [5pts]: Show that the differentiation operator can be written as

$$\hat{\Phi}_k = \frac{\hat{\rho}_k(\Delta x)^2}{2(\cos\frac{2\pi k}{J} - 1)} \tag{12}$$

in Fourier space, i.e. $\hat{\Phi}_k$ and $\hat{\rho}_k$ are the coefficients of the discrete Fourier transform

$$\rho_j = \frac{1}{J} \sum_{k=0}^{J-1} \hat{\rho}_k \, e^{-2\pi\imath jk/J}. \tag{13}$$

(7g) The Poisson Solver [15pts]: Implement the Poisson solver based on eq. 12 in the function `poissonfft` in `pde_poissonfft.py`. Your program should return two plots in one figure: the density distribution $\rho(x)$, and the resulting potential $\Phi(x)$. Your program should also take an optional parameter `Jmax` such that if you specify `pde_poissonfft.py 64 point 512`, the results are overplotted in the same plot, for $J = 64, 128, 256, 512$. That way, you can check the convergence behavior.