PHY 604: Computational Methods in Physics and Astrophysics II

Homework #4

Due: Nov. 14,2023

Programs can be written in any language (but python is recommended), In addition to the program, you should have a writeup that contains the plots requested in the homework questions, answers to any analytical or explanation questions, and a short description of your code and how to run it. This can be done in, e.g., LaTeX, markdown, etc. Combining the code and writeup in jupyter notebooks is highly recommended.

Code and writeup should be submitted using `git` *via github in the repo that was created from* github classroom *link.*

1. *Convolutions.* A convolution is defined as:

$$(f \star g)(t) \equiv \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau \tag{1}$$

It is easy to compute this with FFTs, via the *convolution theorem*,

$$\mathcal{F}\{f \star g\} = \mathcal{F}\{f\}\,\mathcal{F}\{g\} \tag{2}$$

That is the Fourier transform of the convolution of f and g is simply the product of the individual transforms of f and g. This allows us to compute the convolution via multiplication in Fourier space and then take the inverse transform, $\mathcal{F}^{-1}\{\}$, to recover the convolution in real space:

$$f \star g = \mathcal{F}^{-1}\{\mathcal{F}\{f\}\,\mathcal{F}\{g\}\} \tag{3}$$

The file `signal.txt` in your git repo contains data of a function polluted with noise. We want to remove the noise to recover the original function. The three columns in the file are: x, $f^{(\mathrm{orig})}(x)$, and $f^{(\mathrm{noisy})}(x)$.

Consider the following kernels:

- tophat:

$$q^{\mathrm{th}}(x) = \begin{cases} 1/L & \text{if } x < L \\ 0 & \text{if } x \geq L \end{cases} \tag{4}$$

- Gaussian:

$$q^{\mathrm{gauss}}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(x/\sigma)^2} \tag{5}$$

For each of these kernels, plot the FFT (feel free to use a library for the FFT, the code from class, or your own implementation). Compute the convolution of $f^{(\mathrm{noisy})}(x)$ and $q(x)$ in Fourier space and transform back to real space, and plot the *de-noised* function together with the original signal from the `signal.txt` (i.e., $f^{(\mathrm{orig})}(x)$). Experiment with the tunable parameters in the kernels (L or σ) to see how clean you can get the noisy data and comment on what you see.

This process is used a lot in image processing both to remove noise and to compensate for the behavior of cameras to sharpen images.

Note: The Gaussian kernel is centered around $x = 0$; thus, since we require the function to be periodic, it should be split between the beginning and end of the interval (e.g., see Fig. 1). Also, ensure that your kernel's are normalized (i.e., add up to 1 over the domain).

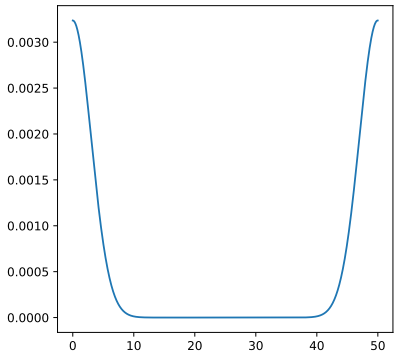

Figure 1: Gaussian kernel, made to be periodic.

2. *Three time-level schemes for the diffusion equation* (based on Garcia)

(**a**) The Richardson scheme for solving the diffusion equation uses the following discretization:

$$\frac{T_i^{n+1} - T_i^{n-1}}{2\tau} = \kappa \frac{T_{i+1}^n + T_{i-1}^n - 2T_i^n}{h^2}. \tag{6}$$

Not that this method is a "three time-level" scheme, so it requires a different scheme to get started. Write a program that uses this scheme to solve the diffusion equation with the initial conditions discussed in Lecture 15 (see `Diffusion_FTCS.ipynb`). Use the FTCS scheme on the first step to get started. Try a few values of τ and show that this method is always unstable.

(**b**) The DuFort-Frankel scheme for solving the diffusion equation uses the following discretization:

$$\frac{T_i^{n+1} - T_i^{n-1}}{2\tau} = \kappa \frac{T_{i+1}^n + T_{i-1}^n - (T_i^{n+1} + T_i^{n-1})}{h^2}. \tag{7}$$

Do the same as part **(a)**, and show that this scheme is unconditionally stable, but it's accuracy decreases with increasing τ.

3. *Other methods for the advection equation* (based on Garcia)

(**a**) The "upwind" scheme for solving the advection equation uses a left derivative for the $\partial/\partial x$ term,

$$\frac{a_i^{n+1} - a_i^n}{\tau} = -c \frac{a_i^n - a_{i-1}^n}{h}. \tag{8}$$

Use this method to solve the advection equation for the initial conditions used in `advect-to-post.ipynb` (Lecture 15). For what values of τ is this method stable?

(**b**) The leap-frog scheme for solving the advection equation uses centered derivatives for both terms:

$$\frac{a_i^{n+1} - a_i^{n-1}}{2\tau} = -c \frac{a_{i+1}^n - a_{i-1}^n}{2h}. \tag{9}$$

Note that this a three time-level scheme, so to get started you need to use one step of a different scheme (e.g., Lax). Use this method to solve the advection equation for the initial conditions used in `advect-to-post.ipynb`. For what values of τ is this method stable?