Math 567 **Homework 2** **Due Sep. 25, 2020**
Grady Wright

1. **(Spectral differentiation)**. As discussed in class, the discrete Fourier series for a 2π periodic function u, sampled at the points $t_j = \frac{2\pi j}{N}$, $j = 0, 1, \ldots, N-1$, where N is odd, is given

$$p_N(t) = \sum_{k=-\frac{N-1}{2}}^{\frac{N-1}{2}} \tilde{c}_k e^{ikt}, \tag{1}$$

where the discrete Fourier coefficients are given as

$$\tilde{c}_k = \frac{1}{N} \sum_{j=0}^{N-1} u_j e^{-\frac{2\pi i}{N} jk}, \; k = -\frac{N-1}{2}, \ldots, \frac{N-1}{2}. \tag{2}$$

Here u_j are the samples of u at t_j. The formula for the coefficients arise from the trapezoidal rule approximation of the continuous Fourier coefficients. Recall that $p_N(t)$ interpolates u_j at t_j.

The forward and inverse discrete Fourier transforms (DFTs) are commonly defined as follows:

$$\text{forward DFT: } \hat{u}_k = \sum_{j=0}^{N-1} u_j e^{-2\pi i jk/N}, \; k = 0, \ldots, N-1, \tag{3}$$

$$\text{inverse DFT: } u_j = \frac{1}{N} \sum_{k=0}^{N-1} \hat{u}_k e^{2\pi i jk/N}, \; j = 0, \ldots, N-1. \tag{4}$$

Letting $\hat{\mathbf{u}}$ denote the row-vector containing the N coefficients $\hat{u}_k$, the discrete Fourier coefficients can be recovered from $\hat{\mathbf{u}}$ as follows:

$$\hat{\mathbf{u}} = \frac{1}{N} \begin{bmatrix} c_0 & c_1 & \cdots & c_{(N-1)/2} & c_{-(N-1)/2} & \cdots & c_{-2} & c_{-1} \end{bmatrix}.$$

See the class notes for the definitions when N is even. The fast Fourier transform (FFT) and its inverse compute (3) and (4), respectively.

In this problem you will be approximating derivatives of the 2π periodic function

$$u(t) = e^{\cos(5(t-0.1))}$$

over $[0, 2\pi)$ using the FFT and finite differences. For all problems, you will be using the sample points $t_j = \frac{2\pi j}{N}$, $j = 0, 1, \ldots, N-1$, for an odd value of N.

(a) Implement a code that uses the FFT and its inverse to approximate $u'(t)$ at the sample points t_j, $j = 0, 1, \ldots, N-1$. Your code should take in the samples u_j and compute the approximate derivatives u'_j using the FFT library that is part of the language you are using:

- In MATLAB you can just call `fft` and `ifft` directly (these use FFTW)
- In Juila you need to add the FFTW package
- In Python you can use the FFT from NumPy, or better yet, use `pyFFTW`.

(b) Using your code from part (a), approximate the derivative of u using $N = 63$ and $N = 127$ and compare this to correct derivative by plotting the difference between the correct derivative and the approximate derivative vs. the sample points t_j, for each value of N. Include plots of the error for both values of N in your write-up.

(c) Compute the approximate derivative of u at t_j using the centered second order accurate finite difference formula

$$u'_j = \frac{1}{2h}\left(-u_{j-1} + u_{j+1}\right),\ j = 0, 1, \ldots, N-1,$$

where $h = 2\pi/N$. In this formula you can use the fact that u is periodic to get values for u_{-1} and u_N:

$$u_{-1} = u_{N-1} \quad \text{and} \quad u_N = u_0.$$

You may wish to construct a sparse differentiation matrix D_N for this computation, which takes the form, when $N = 7$,

$$D_7 = \frac{1}{2h}\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 \end{bmatrix}.$$

The general matrix D_N can be easily constructed using the `toeplitz` functions in MATLAB, Julia, or NumPy. Similar to part (b), compute these approximations using $N = 63$ and $N = 127$ and make plots of the error. Include these plots in your write-up.

(d) Comment on what you notice from the results from (b) and (c) in terms of which technique is more accurate and what happens to the errors when N is increased from 63 to 127.

2. **(Discrete convolution)**. The discrete convolution of two vectors $\mathbf{a} = \{a_j\}_{j=0}^{N-1}$ and $\mathbf{x} = \{x_j\}_{j=0}^{N-1}$ is defined as a new vector $\mathbf{b}$ with entries

$$b_m = \sum_{j=0}^{N-1} a_{m-j} x_j, \quad m = 0, \ldots, N-1, \tag{5}$$

or in matrix-vector form as

$$\underbrace{\begin{bmatrix} a_0 & a_{N-1} & a_{N-2} & \cdots & a_1 \\ a_1 & a_0 & a_{N-1} & \cdots & a_2 \\ a_2 & a_1 & a_0 & \cdots & a_3 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ a_{N-1} & a_{N-2} & a_{N-3} & \cdots & a_0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{N-1} \end{bmatrix}}_{\mathbf{b}} \tag{6}$$

The structure exhibited in A gets a special name in linear algebra: *circulant*. Equation (5) can be viewed as a discretization of the continuous circular convolution (up to scaling) of two T-periodic functions $a(t)$ and $x(t)$ on the interval $[t_0, t_0 + T]$:

$$(a * x)(t) = \int_{t_0}^{t_0+T} a(t-\tau)x(\tau)d\tau, \tag{7}$$

where $a_{m-j} \approx a(t_m - t_j)$, $x_j \approx x(t_j)$, and $t_\ell = t_0 + (T/N)\ell$. Circular convolutions and their discrete versions play a fundamental role in signal processing (your cell phones would not work without them!).

One of the most important results of the discrete Fourier transform (DFT) is that it can be used to relate the discrete convolution of two vectors to the DFT of the individual vectors in a

beautiful way. The result is known as the discrete (or circular) convolution theorem and says the following. Let $\hat{\mathbf{a}}$ and $\hat{\mathbf{x}}$ denote the DFTs of $\mathbf{a}$ and $\mathbf{x}$ in (6), respectively. Then the DFT of the discrete convolution $\mathbf{b}$ of $\mathbf{a}$ and $\mathbf{x}$ is given by

$$\hat{b}_m = \hat{a}_m \hat{x}_m, \quad m = 0, \ldots, N-1, \tag{8}$$

or in matrix-vector form:

$$\underbrace{\begin{bmatrix} \hat{a}_0 & 0 & 0 & \cdots & 0 \\ 0 & \hat{a}_1 & 0 & \cdots & 0 \\ 0 & 0 & \hat{a}_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \hat{a}_{N-1} \end{bmatrix}}_{\hat{A} := \operatorname{diag}(\hat{\mathbf{a}})} \underbrace{\begin{bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \hat{x}_2 \\ \vdots \\ \hat{x}_{N-1} \end{bmatrix}}_{\hat{\mathbf{x}}} = \underbrace{\begin{bmatrix} \hat{b}_1 \\ \hat{b}_1 \\ \hat{b}_2 \\ \vdots \\ \hat{b}_{N-1} \end{bmatrix}}_{\hat{\mathbf{b}}} \tag{9}$$

This theorem means that we can recover the result of the discrete convolution of $\mathbf{a}$ and $\mathbf{x}$ by the following procedure:

i Compute the DFTs of $\mathbf{a}$ and $\mathbf{x}$.

ii Obtain $\hat{\mathbf{b}}$ according to (8).

iii Compute the inverse DFT of $\hat{\mathbf{b}}$ to obtain $\mathbf{b}$ in (5).

By using the FFT to compute the DFT, this procedure requires $O(N \log N)$ operations instead of $O(N^2)$ as formula (6) might suggest.

Problem: Verify numerically that the discrete convolution theorem is correct by computing $\mathbf{b}$ directly using (6) and computing it using the procedure (i)–(iii) above with FFT. Use the following

$$\begin{aligned} a(t) &= \frac{1}{\sigma\sqrt{2\pi}} e^{-0.5\frac{t^2}{\sigma^2}}, \\ x(t) &= (1 + 0.05\cos(16t))\cos(2t), \\ t_\ell &= -\pi + 2\pi\ell/N, \ \ell = 0 \ldots, N-1 \end{aligned}$$

to construct the vectors $\mathbf{a}$ and $\mathbf{x}$. Choose $N = 64$ for the length of the vectors and $\sigma = 0.1$. Report the max-norm of the difference between $\mathbf{b}$ computed from the two methods.

If one scales $\mathbf{b}$ by $2\pi/N$, then this gives the approximation to (7). Scale $\mathbf{b}$ in this fashion and plot it together with $\mathbf{x}$ and use t as the independent variable. Note that the scaled $\mathbf{b}$ is a smoothed out version of $\mathbf{x}$. This is one of the main applications of the discrete convolution theorem in signal processing.

3. *Fast solutions of tridiagonal linear systems.* Consider the tridiagonal linear system of equations

$$\underbrace{\begin{bmatrix} b & a & 0 & \cdots & \cdots & \cdots & 0 \\ a & b & a & \ddots & & & \vdots \\ 0 & a & b & a & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & a & b & a \\ 0 & \cdots & \cdots & \cdots & 0 & a & b \end{bmatrix}}_{} \underbrace{\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix}}_{\mathbf{u}} = \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ \vdots \\ f_{N-2} \\ f_{N-1} \end{bmatrix}}_{\mathbf{f}}, \tag{10}$$

and suppose $u_0 = u_N = f_0 = f_N = 0$, and $b \neq 0$. This system can be solved efficiently using the Crout (or Thomas) algorithm for linear systems. However, in this problem you will show how to solve the system using the Discrete Sine Transform (DST).

(a) Let $\hat{\mathbf{u}}$ and $\hat{\mathbf{f}}$ denote the DST of the vectors $\mathbf{u}$ and $\mathbf{f}$ in (10), i.e.

$$\hat{u}_k = \frac{1}{N} \sum_{j=1}^{N-1} u_j \sin\left(\frac{\pi jk}{N}\right),$$
$$\hat{f}_k = \frac{1}{N} \sum_{j=1}^{N-1} f_j \sin\left(\frac{\pi jk}{N}\right),$$

for $k = 1, \ldots, N-1$. Now, we can express the entries of $\mathbf{u}$ and $\mathbf{f}$ as

$$u_j = 2 \sum_{k=1}^{N-1} \hat{u}_k \sin\left(\frac{\pi jk}{N}\right),$$
$$f_j = 2 \sum_{k=1}^{N-1} \hat{f}_k \sin\left(\frac{\pi jk}{N}\right).$$

Substitute these expressions into the linear system (10) to show that row j of this systems simplifies to

$$\sum_{k=1}^{N-1} \hat{u}_k \left(2a \cos\left(\frac{\pi k}{N}\right) + b\right) \sin\left(\frac{\pi jk}{N}\right) = \sum_{k=1}^{N-1} \hat{f}_k \sin\left(\frac{\pi jk}{N}\right),$$

in the case of $j = 1$ and $j = N-1$ use the fact that $u_0 = u_N = 0$.

(b) Use the results from part (a) to show that the DST of the solution to (10) is given by

$$\hat{u}_k = \frac{\hat{f}_k}{2a \cos\left(\frac{\pi k}{N}\right) + b}, \quad k = 1, \ldots, N-1.$$

(c) Put parts (a) and (b) together to explain how to obtain the solution to (10).

(d) Download the MATLAB codes `dst.m` and `idst.m` on the course Blackboard page for computing the DST and inverse DST (these codes use the FFT to do the computation and are thus 'fast'—although not as fast as they could be since they use complex arithemtic). Use these codes, or translate them into another language, and the procedure outlined in (a)–(c) to solve the linear system (10) for $N = 100$, $a = 1$, $b = -2$, and $f_j = h^2 \tanh\left(4 \sin\left(\pi j/N\right)\right)$, for $j = 1, \ldots, N-1$, where $h = \pi/N$. Make a plot of the solution u.

Note: This gives a second-order finite difference solution to the boundary value problem $u''(x) = \tanh\left(4 \sin\left(x\right)\right)$, $0 \le x \le \pi$, $u(0) = u(\pi) = 0$, at the points $x_j = \pi j/N$, $j = 1, \ldots, N-1$.

4. *Implicit FD methods.* Finite difference (FD) schemes can be made more accurate by adding more nodes to the formulas (i.e. increasing the stencil widths). For example the schemes

$$\frac{1}{h^2} \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} u = f,$$

and

$$\frac{1}{h^2} \begin{bmatrix} -\frac{1}{12} & \frac{4}{3} & -\frac{5}{2} & \frac{4}{3} & -\frac{1}{12} \end{bmatrix} u = f$$

are the respective second and fourth-order centered FD approximations to the 1-D Poisson equation $u''(x) = f$. The problem with increasing the stencil width is that special treatment must be given to points near the boundary. This has the potential for altering the nice banded structure of the resulting linear systems, which can lead to a considerable increase in the computational cost involved in solving these systems. An additional problem, especially for the Poisson equation, is that the discrete maximum principle of the schemes can be lost (as illustrated in the second FD scheme above).

An alternative idea for increasing the order of accuracy of the FD schemes without increasing the stencil width is based on an idea first developed in the 1950s by L. Collatz and originally termed *Mehrstellenverfahren* (however, now the term "implicit", "compact", or "Hermite" is typically used, as in Homework 1). The basic idea is to use information from the underlying differential equation to increase the order. The following is an example of such a scheme for the 3-point, centered, approximation to the 1-D Poisson equation $u''(x) = f(x)$

$$\frac{1}{h^2}\begin{bmatrix} 1 & -2 & 1 \end{bmatrix} u = \frac{1}{12}\begin{bmatrix} 1 & 10 & 1 \end{bmatrix} f\,, \tag{11}$$

which matches the formula (5) from problem 3 of Homework 1. In that homework you also showed that this formula is fourth-order accurate.

Fornberg [1, p. 67] describes the following simple technique for deriving the approximation (11). First note that

$$\frac{1}{h^2}\begin{bmatrix} -\frac{1}{12} & \frac{4}{3} & -\frac{5}{2} & \frac{4}{3} & -\frac{1}{12} \end{bmatrix} u = f + O(h^4). \tag{12}$$

By differentiating both sides of the differential equation with respect to x twice, we have $u''''(x) = f''(x)$. Discretizing this equation with second-order accurate FD formulas for $u''''(x)$ and $f''(x)$ gives

$$\frac{1}{h^4}\begin{bmatrix} 1 & -4 & 6 & -4 & 1 \end{bmatrix} u = \frac{1}{h^2}\begin{bmatrix} 1 & -2 & 1 \end{bmatrix} f + O(h^2). \tag{13}$$

Combining equations (12) and (13) with the linear combination

$$(12) + \frac{h^2}{12}(13)$$

results in precisely the fourth-order accurate scheme (11).

(a) Generalize the technique above to derive an implicit, three-point, fourth-order accurate FD formula for the following BVP:

$$u''(x) + k^2 u(x) = f(x), \quad a \le x \le b$$

which is referred to as the Helmholtz equation and k is interpreted as the 'wave-number'.

(b) Use the implicit formula from part (a) to solve Helmholtz equation for $0 \le x \le 1$, with $f(x) = 1$, $k = 150$, and boundary conditions $u(0) = 1$, $u(1) = 0$. Compare your solution to the exact answer,

$$u(x) = \frac{1}{k^2} + \left(1 - \frac{1}{k^2}\right)\cos(kx) - \left(\frac{1}{k^2} + \left(1 - \frac{1}{k^2}\right)\cos(k)\right)\csc(k)\sin(kx),$$

by plotting relative two-norm of the error in the approximate solution as a function of the grid spacing, h. Use $h = 2^{-\ell}$, $\ell = 7, 8, \ldots, 16$ for this plot. Verify that the convergence is $\mathcal{O}(h^4)$. Also plot the approximate solution for the finest grid. For full credit, use the tridiagonal solver that you developed in problem 3 for this problem.

References

[1] B. Fornberg. *A Practical Guide to Pseudospectral Methods*. Cambridge University Press, Cambridge, 1996.