MAE 250H, Spring 2019
J. D. Eldredge
Homework 1, Due Tuesday, April 9

This homework is designed as a review for those that have taken a previous course on numerical methods and as a quick spin-up for those that have not. I recommend that you find one of several basic texts on numerical methods for background. For example, the book by P. Moin in the course syllabus or K. Atkinson, *An Introduction to Numerical Analysis*, Wiley (the text often used for MAE 182C).

As for all homework problems, you are required to make a short write-up with some discussion of your results for each problem, as well as any relevant plots or tables. Finally, any code written for the problem should be attached to the homework.
I have provided a basic set of files for starting on this homework (and to provide an example of how you might write and test code for any homework or project). You can access this as follows, using `git`:

```
git clone https://github.com/incompressible-flows-spring2019/mae250h_hw1.git
```

This will create a directory called `mae250h_hw1` with stuff in it. If I change anything in that directory on the remote server, you can easily get it by doing `git pull`.

1. **Numerical Integration.** Consider the integral

$$I = \int_0^1 f(x)\,\mathrm{d}x$$

where

$$f(x) = \frac{c_1}{\sqrt{x+\alpha_1}} + \frac{c_2}{(x-\alpha_2)^2+\delta^2} - \pi$$

and the parameters in the integrand are given by $c_1 = 1, c_2 = 100, \alpha_1 = 0.01, \alpha_2 = 0.3, \delta^2 = 0.001$. The exact evaluation of the integral is

$$I = 2c_1\left(\sqrt{1+\alpha_1} - \sqrt{\alpha_1}\right) + \frac{c_2}{\delta}\left[\arctan\left(\frac{\alpha_2}{\delta}\right) - \arctan\left(\frac{\alpha_2 - 1}{\delta}\right)\right] - \pi$$

(a) Write a numerical routine that accepts, as input arguments: the function to be integrated (e.g., in Matlab, one uses the notation `@f` to pass a function called `f`; I provide an example in Julia in the provided HW1 module), the limits of integration and the number of integration steps (N). The output should be the result of the integral. Implement both the trapezoidal rule and Simpson's rule in this routine, with the selection of the scheme given as another input argument.

(b) Use the trapezoidal rule to evaluate the integral, with $N = 16, 32, 64, \ldots$. Evaluate the error for each choice of N. Make a log-log plot of error versus N and verify that the slope is consistent with the order of accuracy of the method.

(c) Do the same as in (b) for Simpson's rule.

2. **Construction of a tridiagonal matrix solver.** Linear systems of equations involving *tridiagonal matrices* arise frequently in CFD. Here, we mean a system

$$Ax = f$$

where A is a $M \times M$ matrix with the form

$$A = \begin{bmatrix} b_1 & c_1 & 0 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & 0 & \cdots & 0 \\ 0 & a_3 & b_3 & c_3 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{M-1} & b_{M-1} & c_{M-1} \\ 0 & \cdots & 0 & 0 & a_M & b_M \end{bmatrix}$$

Note that the subdiagonal (with a_j) and superdiagonal (with c_j) have one fewer entry than the main diagonal with b_j, $j = 1, \ldots, M$. Since this notation is rather cumbersome, a simpler notation is often used: $B(M; \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$, where B stands for *banded*, the first argument is the size of the matrix, and the vector arguments of the three bands are given as the remaining arguments. If the diagonal vectors are constant, i.e. $a_j = a$ for $j = 2, \ldots, M$, $c_j = c$ for $j = 1, \ldots, M-1$ and $b_j = b$, $j = 1, \ldots, M$, then we use the notation $B(M; a, b, c)$. This is usually called a *simple* tridiagonal matrix.

Another matrix that arises frequently, in periodic domains, is the *circulant* matrix, in which every row has the same entries, and each row is shifted one element to the right from the one above it. A circulant tridiagonal matrix is

$$A_p = \begin{bmatrix} b & c & 0 & 0 & \cdots & a \\ a & b & c & 0 & \cdots & 0 \\ 0 & a & b & c & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a & b & c \\ c & \cdots & 0 & 0 & a & b \end{bmatrix}$$

The band notation is given the subscript p (for *periodic*) for this matrix: $B_p(M; a, b, c)$.

When such matrices arise, it is important to have an efficient solver that takes advantage of the simple and sparse structure of the matrices. For circulant tridiagonal matrices, this advantage can be found by exploiting the eigensystem of these matrices, which is particularly simple.

Circulant tridiagonal matrices. First, recall that, if a $M \times M$ matrix A has a complete eigensystem, then it has M linearly independent eigenvectors, v_m, where

$$Av_m = \lambda_m v_m, \quad m = 1, \ldots, M$$

and λ_m is the corresponding eigenvalue. The eigenvectors can form the columns of a matrix, V, whereby the M eigenrelations are compactly written as

$$AV = V\Lambda,$$

where Λ is a diagonal matrix with entries λ_m, for $m = 1, \ldots, M$. By multiplying both sides by the inverse V^{-1}, A can be diagonalized, $\Lambda = V^{-1}AV$.

Because the eigenvectors are linearly independent, they form a basis, and thus any vector can be represented as a linear combination of them. (We assume that the eigenvectors have been normalized so that they have unit length.) We can write the solution vector in this basis as $x = V\hat{x}$, where the entries of vector $\hat{x}$ are the coefficients of x in this basis. Also, the coefficients of the right-hand side vector f in this basis are $\hat{f} = V^{-1}f$. Thus, if we left-multiply both sides of $Ax = f$ by V^{-1} and replace x with $V\hat{x}$, we obtain

$$V^{-1}AV\hat{x} = V^{-1}f \Longrightarrow \Lambda\hat{x} = \hat{f},$$

which can be easily solved for $\hat{x}$ as $\Lambda^{-1}\hat{f}$ (which simply involves dividing entries of $\hat{f}$ by the corresponding eigenvalues). Thus,

$$x = V\Lambda^{-1}V^{-1}f$$

Generally, it is not that helpful to diagonalize a matrix in order to solve a linear system. However, for circulant tridiagonal matrices, the eigenvalues and eigenvectors are already known analytically. Even better, the product of the eigenvector matrix (or its inverse) with a vector can actually be carried out faster than the usual matrix-vector product.

For a circulant tridiagonal matrix, the mth eigenvalue (where $m = 1, \ldots, M$) is

$$\lambda_m = b + (a + c)\cos\left(\frac{2\pi(m-1)}{M}\right) - \mathrm{i}(a - c)\sin\left(\frac{2\pi(m-1)}{M}\right)$$

and the corresponding mth eigenvector has components

$$v_{jm} = \mathrm{e}^{2\pi\mathrm{i}(j-1)(m-1)/M}, \quad j = 1, \ldots M$$

These can be interpreted as the entries in the matrix V, where j is the row and m the column

index. Similarly, the entries in V^{-1} are

$$v'_{jm} = \frac{1}{M} \mathrm{e}^{-2\pi \mathrm{i}(j-1)(m-1)/M}, \quad j = 1, \ldots M$$

Based on this structure, it is important to note that the product $\hat{f} = V^{-1} f$ is actually equivalent to the discrete Fourier transform of f (modulo some constant factor). Thus, instead of the usual $O(M^2)$ number of operations required to carry out this matrix-vector product, only $O(M \log M)$ operations are required if we use a fast Fourier transform (FFT). The product $\hat{x} = \Lambda^{-1} \hat{f}$ only requires M operations (divisions of $\hat{f}$ entries by the eigenvalues). Finally, $x = V\hat{x}$ is the inverse discrete Fourier transform, which also is carried out by FFT in $O(M \log M)$ operations. Thus, the solution of a circulant tridiagonal system can be carried out in $O(M \log M)$ operations.

Regular tridiagonal matrices. For a (simple) regular tridiagonal matrix — in which all entries on each diagonal are the same — the eigensystem is only slightly more complicated, so we might attempt to exploit the same approach as for circulant matrices. The mth eigenvalue ($m = 1, \ldots, M$) is

$$\lambda_m = b + 2\sqrt{ac} \cos\left(\frac{m\pi}{M+1}\right)$$

The jth element of the corresponding eigenvector (and the jm entry in V) are

$$v_{jm} = \left(\frac{a}{c}\right)^{(j-1)/2} \sin\left(\frac{jm\pi}{M+1}\right)$$

The entries of V^{-1} are

$$v'_{jm} = \frac{2}{M+1} \left(\frac{c}{a}\right)^{(m-1)/2} \sin\left(\frac{jm\pi}{M+1}\right).$$

This, too, has a structure that can take advantage of the FFT, because multiplication by V or V^{-1} effectively involves a discrete sine transform, which is intimately related to the discrete Fourier transform. Thus, the solution of a regular tridiagonal system can also be carried out in $O(M \log M)$ steps.

For cases in which $a = c$ or for matrices of modest size, this approach would be fine. However, for large matrices where if $c \neq a$, the factor involving $(c/a)^{(m-1)/2}$ or $(a/c)^{(j-1)/2}$ gets quite large for large index, and round-off errors start to become a problem. So this is not the best approach.

A simpler algorithm, which works for both simple and non-simple regular tridiagonal matrices, is based on a stripped-down version of Gaussian elimination that is specialized to tridiagonal matrices, called the *Thomas algorithm*. It proceeds as shown in algorithm 1. Note that the entries are numbered by the row in which they sit. Thus, b_j, $j = 1, \ldots, M$, describes the

diagonal, and a_j, $j = 2, \ldots, M$ the subdiagonal, and c_j, $j = 1, \ldots, M-1$ the superdiagonal. The algorithm consists of one forward sweep, followed by a back substitution, as in Gaussian elimination.

Algorithm 1 Thomas algorithm

$c'_1 = c_1/b_1$
$f'_1 = f_1/b_1$
for $j = 2 : M-1$
  $c'_j = c_j/(b_j - c'_{j-1}a_j)$
  $f'_j = (f_j - f'_{j-1}a_j)/(b_j - c'_{j-1}a_j)$
end for
$f'_M = (f_M - f'_{M-1}a_M)/(b_M - c'_{M-1}a_M)$
$x_M = f'_M$
for $j = M-1 : -1 : 1$
  $x_j = f'_j - c'_j x_{j+1}$
end for

I have provided a Julia function called `trisolve` in `HW1.jl` (and a Matlab routine with the same name) that carries out the FFT solution for circulant tridiagonal matrices. It accepts as input five arguments: The three diagonal entries (`a`, `b`, `c`), the right-hand side vector `f`, and a `type` argument (a string) that specifies the type of matrix. The Julia function is demonstrated in a Jupyter notebook in the repository.

Your task is to extend this routine to allow solution of *regular* tridiagonal systems, possibly with non-simple diagonals. You will have to implement the Thomas algorithm.

Some notes:

- Clearly, this task also requires that you use an FFT library with your code. In Matlab, this is straightforward (`fft`). For other languages, I suggest you look into the *FFTW* library. In Julia, you can get this by adding the FFTW package in the usual package manager way.
- Part of the goal of this assignment is to pay attention to the code's efficiency. The way that the Thomas algorithm is implemented is crucial in determining the efficiency of your code. For example, if the elements of the arrays are not accessed sequentially in the loops, then there are larger *memory strides*, and you will likely see a large drop in performance. You should use your language's built-in CPU timer routines (see `tic` and `toc` in Matlab, and `@time` macro in Julia) to check the time required to compute the solution of the system, and try to find ways to speed it up. In Matlab, you should explore the `profile` command, which allows you to figure out where your code is spending the most time, and thus try to improve it. The equivalent in Julia is the `Profile` package. You should also compare your performance with simply using the built-in solver, e.g. `x=A\f` in Matlab or Julia.

- You also need to verify that you are getting the correct answer for a variety of different matrices and right-hand side vectors. I suggest choosing a right-hand side given by $f_j = 6\sin(P\pi j/(M+1))$, for $j = 1, \ldots, M$, were P is an integer, and set $a = 1$, $b = 4$ and $c = 1$. This is motivated by a finite-difference formula we will discuss in the next section. For this matrix and right-hand side, your answer should approach $\sin(P\pi j/(M+1))$ as M increases.
- Julia and Matlab are both quite flexible for creating *overloaded* functions that run slightly differently based on the types of arguments passed in. In fact, this is one of Julia's most highly-celebrated features, called *method dispatch*. For example, here we can allow the arguments (`a`, `b`, `c`) to be either scalars or arrays and choose the case appropriately. For circulant matrices there is only one possible case: they must be scalars. But for regular tridiagonal matrices we would want to allow these arguments to possibly be vectors with the appropriate lengths ($M-1$ for `a` and `c` and M for `b`). In Matlab, the function `isscalar` is useful to test what type of argument has been passed. For other languages, one can also create overloaded functions and I suggest that you try to do so here. But you can also just create different routines for the simple and non-simple cases.
- In Julia, for testing your result with `x=A\f`, you can create a tridiagonal matrix with the `Tridiagonal` function in the `LinearAlgebra` package. A circulant matrix can be produced with `Circulant` function in the `ToeplitzMatrices` package. In Matlab, for testing your result with `x=A\f`, you can create tridiagonal and circulant matrices with the `gallery` command, with the options `'tridiag'` and `'circul'`, respectively.