M3/4/5N9 – Project 1
Due 28 Nov 2016

Suppose one would like to generate an approximate numerical solution to the differential equation

$$-\frac{d^2u}{dx^2} = f(x) \tag{1}$$

for $u(x)$ on the domain $x \in [0, 2\pi)$ subject to periodic boundary conditions $u(0) = u(2\pi)$ and a zero-mean condition, $\int_0^{2\pi} u(x)dx = 0$. The function $f(x)$ is a prescribed 2π-periodic function with zero-mean.

One way to do this is to replace the differential operator by a finite difference operator that gives approximations to d^2u/dx^2 at the equispaced points $x_j = (j-1)\Delta x$ for $j = 1, \ldots, N$, where $\Delta x = 2\pi/N$. One example is the second-order central difference scheme

$$\left.\frac{d^2u}{dx^2}\right|_{x=x_j} \approx \frac{u_{j+1} - 2u_j + u_{j-1}}{\Delta x^2} \tag{2}$$

where u_j is the approximate solution at x_j. If we apply the periodic conditions by requiring that $u_1 = u_{N+1}$, the approximation of second derivative operator for a periodic function is the $N \times N$ matrix

$$D_2 = \frac{1}{\Delta x^2}\begin{bmatrix} -2 & 1 & & & 1 \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ 1 & & & 1 & -2 \end{bmatrix} \tag{3}$$

To enforce the zero-mean condition, $\sum_{i=1}^{N} u_i = 0$, we replace each element of the first row of D_2 by 1. Putting this all together and rescaling the first row, we generate the numerical solution, u_N, by solving the linear system

$$Au_N = f_N \tag{4}$$

where we have the matrix

$$A = \frac{1}{\Delta x^2}\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ -1 & & & -1 & 2 \end{bmatrix} \tag{5}$$

and the vectors are $u_N = [u_1, u_2, \ldots, u_N]^T$, and $f_N = [0, f(x_2), \ldots, f(x_N)]^T$.

Questions

1. Solve the linear system for u_N using QR with $N = 16, 32, 64$ and 128 and $f(x) = \sin x$. When you do this, use the code `house` (on Blackboard) to factor A and write the additional codes that are needed to perform the remaining steps of the algorithm. Make a table of the error,

$$E_N = \sqrt{\Delta x \sum_{j=1}^{N} \left(u(x_j) - u_{N;j}\right)^2} \tag{6}$$

where $u(x) = \sin x$ is the exact solution. If your code is working properly, you should observe that the error decreases $\sim \Delta x^2$, especially for larger values of N.

2. Show that vectors $z_k = (1/\sqrt{N})[e^{i(k-1)x_1}, e^{i(k-1)x_2}, \ldots, e^{i(k-1)x_N}]^T$ for $k = 1, \ldots, N$ are eigenvectors of the matrix D_2. Find the corresponding eigenvalues, λ_k, for each k. In particular, show that for eigenvector $z_1 = (1/\sqrt{N})[1, 1, \ldots, 1]^T$, we have $\lambda_1 = 0$.
   Hint: Use Eq. (2).

3. Show that the eigenvectors form an orthonormal set with respect to the standard inner product, $\langle u, v \rangle = u^* v$.

4. The matrix D_2 can be decomposed such that $D_2 = Z \Lambda Z^{-1}$ where Z is the matrix whose columns are the eigenvectors z_k and Λ is the diagonal matrix with $\Lambda_{kk} = \lambda_k$. Starting with this decomposition, write and implement in MATLAB an algorithm to solve the linear system Eq. (4). In your algorithm, you may use the operations adjoint, *, matrix-vector multiplication, and division, $/$, but certainly not backslash! Demonstrate that your implementation is working correctly by reproducing your results from part 1.
   Hint: Use the inner product of u_N with z_1 to satisfy the zero-mean condition.

5. Carefully state the backward stability conditions that need to be satisfied by each step of your algorithm in part 4. From lectures we know that the QR algorithm used in part 1 and the basic floating point operations are backward stable. Based on these results, provide an explanation as to why the algorithm for part 4 should also be backward stable.

6. Obtain operation counts for the algorithms used in parts 1 and 4. Show that the algorithm in part 4 is much faster than using QR as $N \to \infty$ and provide an explanation as to why this is the case. Use the MATLAB functions `tic` and `toc` to measure runtimes and confirm your analysis by plotting the runtime vs. N on log-log scale for both approaches.

7. It turns out that the algorithm in 4 can be made even faster using the *fast Fourier transform* and its inverse to perform *implicitly* the following matrix-vector multiplications

$$U_k = \sum_{j=1}^{N} u_j e^{-i(k-1)(j-1)\Delta x} \tag{7}$$

$$u_j = \frac{1}{N} \sum_{k=1}^{N} U_k e^{i(k-1)(j-1)\Delta x} \tag{8}$$

   in $O(N \log N)$ operations as $N \to \infty$. In MATLAB, these computations correspond to calling the functions `U = fft(u)` and `u = ifft(U)`. State which matrix-vector multiplications used in part 4 can be replaced by these function calls and modify your code to incorporate these changes. Check that your numerical solutions are still identical to those produced in parts 1 and 4. Use `tic` and `toc` to measure runtimes for different values of N and compare these values with those obtained in part 6. Do you observe a speed up in the computations?