CS556 Iterative Methods Fall 2024 Homework 2.

Due Tuesday, Sept. 24, 5 PM.

1a. Consider $A\underline{u} = \underline{f}$, where A is the $n \times n$ SPD matrix derived from the 2nd-order centered difference approximation to $-\nabla^2 u = f$ with homogeneous Dirichlet conditions on the d-dimensional unit cube, $\Omega = [0,1]^d$. Assume a uniform spacing $h = 1/(m+1)$ in each direction (implying $n = m^d$).

Suppose we use Jacobi iteration to solve this system, starting with $\underline{u}_0 = 0$,

$$\underline{u}_k \;=\; \underline{u}_{k-1} \,+\, D^{-1}(\underline{b} \,-\, A\underline{u}_{k-1}),$$

where $D = \text{diag}(a_{ii})$ is the diagonal of A. The error propagator for this system is $E = (I - D^{-1}A)$ and it has a spectral radius of the form

$$\rho(E) \;=\; 1 \,-\, \epsilon,$$

with $\epsilon \sim Cn^k$. Find C and k in this expression for $d = 1$, 2, and 3.*

1b. Use the results

$$[\rho(E)]^k \;\sim\; (1-\epsilon)^k \;\sim\; (e^{-\epsilon})^k \;\approx\; 10^{-\frac{\epsilon k}{2}}$$

to derive an expression for the anticipated number of iterations for the relative error, $\|\underline{e}_k\|/\|\underline{u}\| \le 10^{-6}$ for each case, $d = 1$, 2, and 3. (For purposes of this assignment, you can assume that the majority of the energy in the solution is in the most slowly decaying mode.)

2. Using material we've covered in class to date, complete the table below for the class of problems described in **1**. Where possible, give the asymptotic constant or a close approximation, rather than just $O(n^\gamma)$ for some particular γ. Use a relative error bound of $\approx 10^{-6}$ when considering iterative methods.

Computational Complexities

Method	1D flops	1D storage	2D flops	2D storage	3D flops	3D storage
Banded Solver	8n	4n (LU)				
Nested Diss.						
Fast Diag. Meth.						
FFT-based FDM						
Jacobi Iteration						

3. For each of the cases below, plot the requested data as *symbols*, not lines. Then, plot a line of the form $y = \alpha\, n^\beta$ that goes through the set of observed data for the large values of n (where we expect/hope that the asymptotic model holds).

*Recall that $\epsilon \sim Cn^k$ implies that $\lim\limits_{n \longrightarrow \infty} \epsilon = Cn^k$.

3a. Solve the d-dimensional Poisson problems of the preceding question using Gaussian elimination.[†] Specifically, use a lexicographical ordering for the rows and columns of A. For example, the vector of unknowns in the 3D case would be

$$[u_1\, u_2\, u_3\, \cdots\, u_l\, \cdots\, u_n]^T \quad = \quad [u_{111}\, u_{211}\, u_{311}\, \cdots\, u_{ijk}\, \cdots\, u_{mmm}]^T. \tag{1}$$

For the direct method, you will need to form A. The easiest way to do so is (e.g., in 3D) to set $A = I_1 \otimes I_1 \otimes A_1 \,+\, I_1 \otimes A_1 \otimes I_1 \,+\, A_1 \otimes I_1 \otimes I_1$, where I_1 is the $m \times m$ identity matrix and A_1 is the standard tridiagonal SPD operator for the 1D Poisson problem. Make certain that I_1 and A_1 are declared as *sparse* matrices so that the (very large) matrix A will also be sparse.

In matlab, the 3D A matrix can be formed as:

```
e=ones(m,1);
Ax=spdiags([e -2*e e], -1:1, m, m);
dx = 1./(m+1);
Ax = -Ax./(dx*dx); Ix = speye(m);
A2 = kron(Ix,Ax) + kron(Ax,Ix);
A  = kron(Ix,A2) + kron(Ax,kron(Ix,Ix));
```

For $d = 1$, 2, and 3, consider a sequence of problem sizes, $m = \lfloor 2^{\frac{k}{2}} \rfloor$, for $k = 1, 2, 3, \ldots, k_{\max}$. Measure the time t (seconds) required to compute the LU factorization of A for each (k, d) pair and, for $d = 1$ plot t vs. n. In a different color, plot the results for $d = 2$ on the same graph, and again use a third color to add the results for $d = 3$. For the 3D case, just use $m = 1, 2, 3, 4, \ldots, 20$, but go higher if you can, so that you can better understand the asymptotic behavior.

For each space dimension, take $k_{\max}$ to be large enough that $n = 8000$ or more. Note: I suggest to *not* try to do all space dimensions in a single run because the required values of m are quite different. Also, don't take a very large value of $k_{\max}$ initially—work your way up to tolerably large values until everything is working in your code. Most of the runtime ends up being spent on the case $k = k_{\max}$.

In matlab, the timing would look something like:

```
t0=tic;              %% Warm-start
[L,U]=lu(A);
elapsed1(k) =toc(t0);

t0=tic;              %% Actual time
[L,U]=lu(A);
elapsed2(k) =toc(t0);
mflops(k)   = (flops/elapsed2)/1.e6;

disp([k m n elapsed1(k) elapsed2(k) mflops(k)])
```

The warm start is designed to preload L and U so that you're not measuring overhead associated with memory allocation. The time you plot would be `elapsed2()`. Here, `flops`, would be the estimated number of operations to perform the LU decomposition, from your table of question 2.

[†]**Note:** to force the codes to solve the system without re-ordering, we will actually time the operation `LU=lu(A)`, rather than the time for solution of $A\underline{u} = \underline{b}$.

3b. Solve the d-dimensional Poisson problems of the preceding question using Jacobi iteration. Set the relative tolerance to $tol = 10^{-6}$ and the maximum iteration count to $i_{\max} = 10^6$. Don't bother timing cases for any value of $n > n_{fail}$, where n_{fail} is the size of the first problem where the relative residual norm is $> tol$ after $i_{\max}$ iterations. Make a plot similar to that for **3a**, with time on the y axis and n on the x axis of a loglog plot. *Add to this plot a plot of iteration counts, using the same colors as for* $d = 1$, *2 and 3, but a different symbol than used for the timing.*

3c. Solve the d-dimensional Poisson problems of the preceding question using the fast diagonalization method (FDM). Make a similar plot with three graphs, one for each space dimension. I suggest you form the scaled eigenvector matrix explicitly, rather than by calling `eig()`. As a reminder, the 1D matrix of orthonormal eigenvectors can be generated as

```
i=[1:m]';
ij=i*i';
h     = 1./(m+1);
scale = sqrt(2*h);
S = scale*sin(ij*(pi/(m+1));
```

Verify that this S satisfies two properties:

- $S^T S = I$
- $S^T A S = \Lambda$

where Λ is the matrix of eigenvalues, $\lambda_k = \frac{2}{h^2}\left(1 - \cos\frac{\pi k}{m+1}\right)$.

Note, one should nominally count the construction of S in the "solve" time. (Really, it's part of the "factor" time.) You may choose to do so, or you can leave it out. In the *important* 3D case, the setup time for S is negligible, even for the general case where we must use `eig()` to find the eigenvalues and eigenvectors.

4. Discuss the observations from your plots of question **3**. Specifically,

- *Do your observed timings match the expected complexity estimates of part* **2***?*
- *If not, what might be the cause for the discrepancy?*
- *Which solution strategy is fastest?*
- *How does dimensionality, d, play a role in choosing a solver?*

Pay particular attention to the last of these questions.