CS556 Iterative Methods Homework 2.

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.*

ANS. For this problem, $D = \delta I$ with

$$\delta \;=\; \frac{2d}{\Delta x^2} \;=\; 2dN^2,$$

where $\Delta x = 1/N$ and the number of unknowns $n = (N-1)^d \sim N^d$.
For $d = 3$, the eigenvalues of A are

$$\lambda_{ijk} \;=\; \lambda_i \;+\; \lambda_j \;+\; \lambda_k,$$

with a similar expression in 2D. Here, the λ_i are the eigenvalues of the 1D problem,

$$\lambda_i \;=\; \frac{2}{\Delta x^2}[1 \;-\; \cos(\pi \Delta x i)],$$

Thus, in d space dimensions the extreme eigenvalues of A are

$$\begin{aligned}
\lambda_{\min} &= \frac{2d}{\Delta x^2}[1 \;-\; \cos(\pi \Delta x)] \\
\lambda_{\max} &= \frac{2d}{\Delta x^2}[1 \;-\; \cos(\pi (N-1) \Delta x)].
\end{aligned}$$

For Jacobi iteration, the error propagation matrix $E = I - D^{-1}A$ has eigenvalues $\mu = 1 - \delta\lambda$. Given the eigenvalues of A above, we have that $\mu_{\max} \;=\; -\mu_{\min}$, so $\rho(E) = \mu_{\max}$,

$$\begin{aligned}
\rho(E) &= 1 \;-\; \frac{\Delta x^2}{2d}\frac{2d}{\Delta x^2}[1 - \cos(\pi \Delta x)] \;=\; \cos(\pi \Delta x)] \\
&\sim 1 \;-\; \frac{(\pi \Delta x)^2}{2} \;=\; 1 \;-\; \frac{\pi^2}{2} N^{-2} \;\sim\; 1 \;-\; \frac{\pi^2}{2} n^{-\frac{2}{d}}.
\end{aligned}$$

Consequently, $C \;=\; \pi^2/2$ and $k \;=\; -\frac{2}{d}$, d=1, 2, and 3.

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

1b. Use the results

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

to derive an expression for the anticipated number of iterations for the relative error, $\|\underline{e}_k\|/\|\underline{u}\| \leq 10^{-6}$ 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. *See (16) in* `tridiag_example.pdf` *on the Relate page.*)

ANS. Since $\underline{e}_k = E^k \underline{e}_0 = E^k \underline{u}$, we have

$$\frac{\|\underline{e}_k\|}{\|\underline{u}\|} \quad \leq \quad [\rho(E)]^k \;\approx\; 10^{-\frac{\epsilon k}{2}} \;\approx\; 10^{-6}. \tag{1}$$

Equating the exponents on the right of the preceding expression leads to

$$\begin{aligned} k \quad &\approx \quad \frac{12}{\epsilon} && (2)\\ &\approx \quad \frac{24}{\pi^2} n^{\frac{2}{d}} && (3)\\ &\approx \quad 2.4n^2 \quad d = 1 && (4)\\ &\approx \quad 2.4n, \quad d = 2 && (5)\\ &\approx \quad 2.4n^{\frac{2}{3}}, \quad d = 3. && (6) \end{aligned}$$

Note that the total work in each case is approximately $4dn$ ops per iteration, which gives the following work complexity estimates:

$d = 1$	$W \sim 10n^3$
$d = 2$	$W \sim 20n^2$
$d = 3$	$W \sim 30n^{\frac{5}{3}}$

Clearly, for a given *number of unknowns*, n, Jacobi iteration is highly unattractive for 1D problems, but must more feasible in higher space dimensions. This trend constrasts sharply with direct solvers for lexicographically-ordered systems of this form. In that case, the factor cost scales as $\sim 2nb^2$ and the solve cost (backward and forward substitution) as $\sim 4nb$, for matrix bandwidth $b = m^{d-1} \sim n^{\frac{d-1}{d}}$. The corresponding complexities for the factor phase are tabulated below.

$d = 1$	$W_F \approx 4n$
$d = 2$	$W_F \sim 2n^2$
$d = 3$	$W_F \sim 2n^{\frac{7}{3}}$

We see that for 3D, even our simple Jacobi iteration has a better complexity (albeit with a relatively large constant) than its direct-solve counterpart. On the other hand, things are relatively balanced for 2D, save that the constant for the direct solve is better (likely *much* better, considering that it is implemented in BLAS3, which attains 1-2 orders of magnitude performance boost over BLAS2 on modern CPUs).

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)	$2n^2$	$2n^{\frac{3}{2}}$	$2n^{\frac{7}{3}}$	$2n^{\frac{5}{3}}$
Nested Diss.	8n	4n	$19.07n^{\frac{3}{2}}$	$\frac{31}{8}n\log_2 n$	$O(n^2)$	$O(n^{\frac{4}{3}})$
Fast Diag. Meth.	$4n^2$	n^2	$8n^{\frac{3}{2}}$	$2n$	$12n^{\frac{4}{3}}$	n
FFT-based FDM	$O(n\log n)$	$3n$	$O(n\log n)$	n	$O(n\log n)$	n
Jacobi Iteration	$10n^3$	$5n$	$20n^2$	$7n$	$30n^{\frac{5}{3}}$	$9n$

Remarks concerning the table: The nested dissection complexities came from the source material referenced in the short set of notes on the topic that are posted on Relate. For the FDM, I did not count the preprocessing costs of solving the eigenproblems. For our particular case, we have a closed-form expression for $S = S_x$, etc., given by $S_{ik} = C\sin\pi ik/(m+1)$ where $C = \sqrt{2/(m+1)}$.

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).

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

$$[u_1\ u_2\ u_3\ \cdots\ u_l\ \cdots\ u_n]^T \;=\; [u_{111}\ u_{211}\ u_{311}\ \cdots\ u_{ijk}\ \cdots\ u_{mmm}]^T. \qquad (7)$$

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));
```

†**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 $A\underline{u} = \underline{b}$.

For $d = 1$, 2, and 3, consider a sequence of problem sizes, $m = \lfloor 2^{\frac{k}{2}} \rfloor$, for $k = 1$, 2, 3,...,$k_{\max}$. 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,...,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.

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]';
```

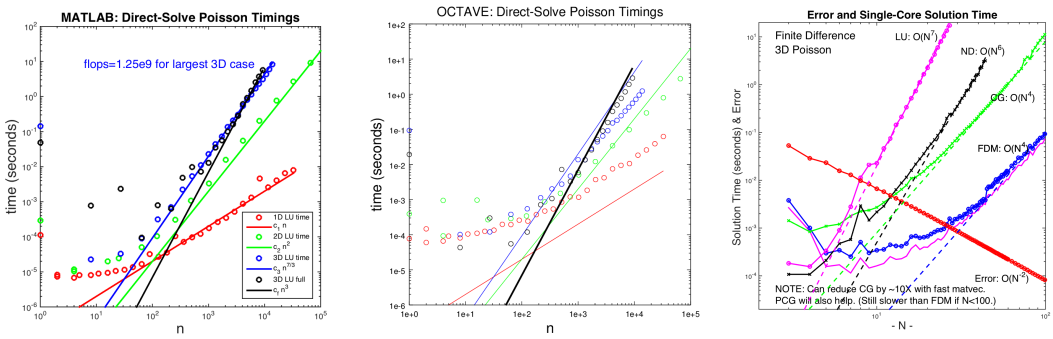

Figure 1: Matlab and Octave timings for direct solver on MacBook Pro (Intel I7).

```
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.

ANS: Figure 1 shows the timings using direct solvers on my Macbook Pro (2017 Intel i7). In addition to 1D, 2D, and 3D, I've put the factor times for a full $n \times n$ matrix. The solid lines are the model fits for Matlab and they have the correct exponent. Surprisingly, Octave outperforms Matlab on my Mac. (The solid lines on the Octave plot are from the Matlab fits.) When I run Matlab, it shows 400% utilization, which I assume means that all 4 cores are running flat out. When I run Octave, it shows 750% utilization – clearly, the vector registers are being heavily used.

The plot on the right compares several solvers for the 3D case only. (Apologies, I haven't yet organized all the runs for the lower-dimensional problems.) From this plot we see that FDM is definitely the fastes on my Mac - even beating conjugate gradient iteration, which outperforms Jacobi. Note that the nested dissection (ND) results here were in fact based on a symmetric approximate minimal degree ordering (SYMAMD in Matlab), which has complexity similar to ND.

I will update these plots in the near future.

Regarding the summary questions below, the question as to which is the fastest approach depends first and foremost on the dimension of the problem and is independent of n, for even modestly large n. FDM is fine for the constant-coefficient case, or under suitable conditions of separability of the governing PDE. It can potentially be used as a preconditioner in the variable coefficient case. Iterative methods might ultimately prevail, especially with projection (e.g. PCG), polynomial accelerators, and preconditioners. In 1D, it will be difficult to beat a direct tridiagonal solve, even in

a parallel computing context. 2D is the contest battleground, for which I would give a slight edge to direct methods, particularly if one considers SYMAMD or ND orderings.

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.