CS 475/675 Spring 2025: Crowdmark Assignment 2

Due May 30, at 11:59 pm Eastern.
Submit all components of your solutions (written/analytical work, code/scripts, figures, plots, output, etc.) to CrowdMark in PDF form in the appropriate section for each question.

You must also separately submit a single zip file containing any and all code/scripts you write to the Crowdmark Assignment 2 DropBox on LEARN, in runnable format (that is, .m).

For full marks, be sure to show all of your work!

1. Let $A = \begin{bmatrix} 2 & 4 & 10 \\ -2 & -5 & -8 \\ 4 & 6 & 27 \end{bmatrix}$.

[6] (a) Compute the LU factorization of A, in which L is unit lower triangular. Do this computation by hand, not in Matlab.

[3] (b) Compute the LDM^T factorization of A. Do this computation by hand, not in Matlab. You should use your answer from part 1a.

[6] (c) Suppose that an arbitrary square matrix, B, has two LDM^T factorizations, i.e. suppose that

$$L_1 D_1 (M_1)^T = B = L_2 D_2 (M_2)^T,$$

where all of L_1, L_2, M_1, M_2 are unit lower triangular, and D_1, D_2 are diagonal, with all their diagonal entries non-zero. Prove that $L_1 = L_2, D_1 = D_2$ and $M_1 = M_2$. (**Remark:** This proves that the LDM^T factorization is unique.)

Hint: Use the theorem that says that if B has two LU factorizations,

$$L_1 U_1 = B = L_2 U_2,$$

in which L_1, L_2 are unit lower triangular and U_1, U_2 are upper triangular, with all their diagonal entries non-zero, then $L_1 = L_2$ and $U_1 = U_2$ (i.e. the LU factorization, in which the L is unit lower triangular and the U has non-zero diagonal entries, is unique).

[8] 2. The following algorithm, to carry out Gaussian Elimination on an $n \times n$ matrix A, is reproduced from the Lecture Notes and Slides.

GE Algorithm

```
for i = 1, 2, ..., n-1
    for k = i+1, ..., n
        mult = a_ki / a_ii
        a_ki = 0                          not needed, but helpful for intuition
        for j = i+1, ..., n
            a_kj = a_kj − mult ∗a_ij      update row k
        end
        b_k = b_k − mult ∗b_i             update RHS
    end
end
```

Compute the count of the flops required to execute this algorithm, in terms of n.

You may use, without proof, the sums:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \text{ and}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}.$$

3. Let $A = \begin{bmatrix} 9 & -12 & 3 \\ -12 & 17 & -4 \\ 3 & -4 & 5 \end{bmatrix}$. It is clear that A is symmetric.

[4] (a) Verify that A is positive definite. You may use Matlab to do this. Whichever way you do your verification, show all of your work.

[8] (b) Compute the Cholesky factor, G, of A. Do this computation by hand, not in Matlab.

4. Let $A = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$. It is clear that A is not symmetric.

[4] (a) Prove that, for any $\vec{0} \neq x \in \mathbb{R}^2$, $x^T Ax > 0$.

[4] (b) Prove that there cannot exist any lower triangular 2×2 real matrix G, with strictly positive entries on its diagonal, such that $GG^T = A$.

5. As in Lecture 04, in modelling (steady state) heat flow, the temperature $T = T(x, y)$ satisfies the Poisson equation

$$-\frac{\partial^2 T}{\partial x^2} - \frac{\partial^2 T}{\partial y^2} = f(x, y)$$

where $f(x, y)$ represents the heat source function.

We approximate $T(x, y)$ at discrete locations on a two dimensional grid with m active grid points in the horizontal dimension and $2m + 1$ active grid points in the vertical dimension (imagine a rectangular metal plate which is 1 unit wide and 2 units tall). Let the (i, j) grid point have location (x_i, y_j) where $x_i = ih$, $y_j = jh$, $h = \frac{1}{m+1}$ is the grid spacing, $0 \le i \le m+1$ and $0 \le j \le 2m+2$. If we let $T_{i,j} \approx T(x_i, y_j)$, then the finite difference approximation results in a set of linear equations

$$\frac{1}{h^2}\left(4T_{i,j} - T_{i-1,j} - T_{i+1,j} - T_{i,j-1} - T_{i,j+1}\right) = f_{i,j}. \tag{1}$$

We will assume that all the boundary temperatures along the sides of the grid are zero:

$$T_{0,j} = T_{i,0} = T_{m+1,j} = T_{i,2m+2} = 0$$

for all i, j as above. We want to analyze the heat flow with two heat sources as given by:

$$f_{i,j} = \begin{cases} 1 \text{ if } \|(x_i, y_j) - (0.3, 1.75)\| \le 0.1 \\ 2 \text{ if } \|(x_i, y_j) - (0.7, 0.25)\| \le 0.2 \\ 0 \text{ otherwise.} \end{cases}$$

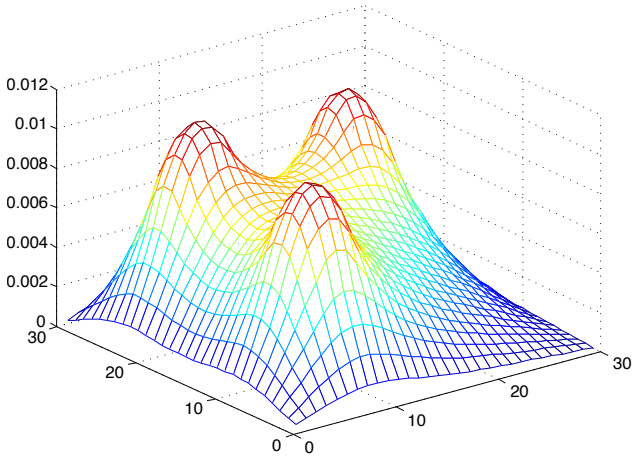

Figure 1: Example plot for three heat sources

An example of a final temperature distribution for a **different** set of three heat sources is shown in Figure 1.

Define a vector x such that

$$x_k = T_{i,j},$$

where $k = i + (j - 1)m$. Similarly, define the vector b with $b_k = f_{i,j}$. Then we can write equation (1) in matrix form as

$$Ax = b. \tag{2}$$

The coefficient matrix A is sparse with size $n \times n$, where $n = m(2m + 1)$.

[10] (a) Create a Matlab function `[A,b] = Lap2D(m)`. The input is a positive integer m, the number of active grid points in the horizontal dimension, as defined above. The outputs are the matrix A and the vector b as in equation (2).

[18] (b) Implement the following numerical methods: Gaussian elimination, Cholesky factorization, and Banded Gaussian elimination. Create the following Matlab functions:

```
x=GaussElim(A,b)
x=Cholesky(A,b)
x=BandGE(A,b,p,q)
```

These Matlab functions take as inputs the matrix A and right-hand side b and compute the solution x using the corresponding variant of Gaussian elimination. The parameters `p` and `q` indicate the lower and upper matrix bandwidth, respectively. (There is no need to implement pivoting or check for zero or small pivots.) You can check your code by visualizing the solution x. Use the following command:

```
>> mesh(reshape(x,m,2*m+1))
```

This will convert the solution vector x into a 2D array, and generate a 2D mesh plot of the result. Submit a 2D mesh plot of the solution for the case when $m = 20$.

[12] (c) Create a Matlab script, `GETimes.m`, that solves (2) using the three different variants of Gaussian elimination you wrote. In particular, set up the matrix A and right-hand side b by calling the Matlab function `Lap2D`. Then solve the equation by calling one of the Matlab functions in part (b). Record the execution time using the Matlab commands `tic` and `toc`. Construct a table of execution times for solving with each of the methods above. Try the values $m = 8, 16, 24, 32$. (Consider vectorizing the innermost loop of your implementations to keep the solve times more manageable.) Compare and comment on the timing results for the three methods you implemented. Include a mesh plot of x, a table of CPU times, and your comments on the timing results.

Submit to the LEARN Dropbox: `Lap2D.m`, `GaussElim.m`, `Cholesky.m`, `BandGE.m`, `GETimes.m`.