CS 475/675 Spring 2025: Crowdmark Assignment 3

Due June 16 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 section for each question.

You must also separately submit a single zip file containing any and all code/scripts you write to the A03 DropBox on LEARN, in runnable format (that is, .m).

You might find these variants of the **triangle inequality** handy for Q3 and/or Q4:

$$\begin{aligned} |a| - |b| &\leq |a + b| \leq |a| + |b|, \\ |a - b| &\geq |a| - |b|. \end{aligned}$$

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

[10] 1. **Column-oriented backward solve.** Consider solving a system $Ux = b$ where U is upper triangular. One approach to its solution is the following process. First, resolve x_n. It can then be removed from equations 1 through $n-1$, and we can proceed with the reduced system in which the right-hand-side has been updated. We next compute x_{n-1} and remove it from equations 1 through $n-2$, etc. For example, if this approach is applied to

$$\begin{bmatrix} 3 & 1 & -1 \\ 0 & 7 & -2 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -2 \\ 17 \\ 4 \end{bmatrix} \tag{1}$$

we first compute $x_3 = 2$, and then solve the reduced 2×2 system:

$$\begin{bmatrix} 3 & 1 \\ 0 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -2 \\ 17 \end{bmatrix} - 2 \begin{bmatrix} -1 \\ -2 \end{bmatrix} \tag{2}$$

Derive an efficient algorithm for the general case where the matrix U is an $n \times n$ upper triangular matrix (not necessarily unit diagonal). Present your algorithm in pseudocode. The inputs to your algorithm should be the matrix U and the right-hand side b, and the output should be the solution x. What is the complexity of your algorithm? Show your work, specifically the coefficients for all $O(n^p)$ terms, $p \in \mathbb{N}$.

[10] 2. Consider an $n \times n$ matrix, A, with upper and lower bandwidths equal to q. Suppose you are given the L and U matrices comprising the LU decomposition of A, i.e., $A = LU$. Derive an **efficient** algorithm (i.e., forward and backward solves) to solve $Ax = b$. Give precise pseudocode for your algorithm. What is the complexity of your algorithm, in terms of n and q? What is the coefficient of the leading order term (assuming $q \gg 1$)?

[10] 3. Let $a_{ij}^{(k-1)}$ be the entries of A after $(k-1)$ steps of Gaussian elimination. Suppose $A^{(k-1)}$ is **strictly column** diagonally dominant; i.e.

$$\left|a_{ii}^{(k-1)}\right| > \sum_{j\geq k, j\neq i} |a_{ji}^{(k-1)}| \qquad i = k, \ldots, n.$$

If Gaussian elimination without pivoting is used; i.e.

$$a_{ji}^{(k)} = a_{ji}^{(k-1)} - \frac{a_{jk}^{(k-1)} a_{ki}^{(k-1)}}{a_{kk}^{(k-1)}} \qquad j = k+1, \ldots, n, \;\; i = k+1, \ldots, n,$$

prove that

$$\left|a_{ii}^{(k)}\right| - \sum_{j\geq k+1, j\neq i} \left|a_{ji}^{(k)}\right| > 0 \qquad i = k+1, \ldots, n.$$

i.e. the submatrix $A^{(k)}$ is also column diagonally dominant. (Hint: pay careful attention to the precise range of the index j. Use the fact that: $\sum_{j\geq k+1, j\neq i} \left|a_{ji}^{(k)}\right| = \sum_{j\geq k+1, j\neq i} \left|a_{ji}^{(k-1)} - \frac{a_{jk}^{(k-1)} a_{ki}^{(k-1)}}{a_{kk}^{(k-1)}}\right|$ and then apply the triangle inequality.)

4. Let A be a strictly **row** diagonally dominant matrix; i.e. A's entries a_{ij} satisfy

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| .$$

[3] (a) As we did in the lectures, let $A = D - L - U$, where D is the diagonal, $-L$ contains only the entries below the diagonal of A, and $-U$ is the entries above the diagonal. Let G be the iteration matrix of the Gauss-Seidel method.
Suppose x and y are vectors such that $y = Gx$. Show that $(D - L)y = Ux$.

[4] (b) Writing out the expression in componentwise form yields

$$a_{jj}y_j + \sum_{i<j} a_{ji}y_i = -\sum_{i>j} a_{ji}x_i. \tag{3}$$

Recall that $\|y\|_\infty = \max_j |y_j|$, and consider the row j corresponding to the largest magnitude entry. Show that the absolute value of the left side of the equality (3) above satisfies

$$\left| a_{jj}y_j + \sum_{i<j} a_{ji}y_i \right| \geq \left(|a_{jj}| - \sum_{i<j} |a_{ji}| \right) \|y\|_\infty.$$

[4] (c) Suppose now that x is a vector with infinity norm of 1. Considering the right side of the earlier equality (3), show

$$\left| \sum_{i>j} a_{ji} x_i \right| \leq \left(\sum_{i>j} |a_{ji}| \right) .$$

[4] (d) Parts (b)-(c) together imply that

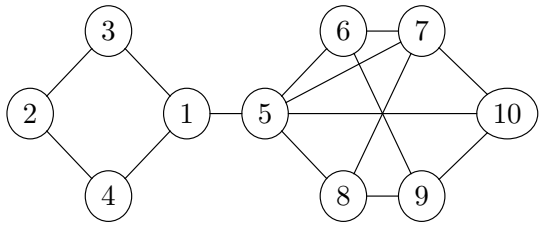
$$\left(|a_{jj}| - \sum_{i<j} |a_{ji}| \right) \|y\|_\infty \leq \left(\sum_{i>j} |a_{ji}| \right).$$

Recalling that an equivalent definition of the matrix infinity norm is

$$\|G\|_\infty \equiv \max_{\|x\|_\infty = 1} \|Gx\|_\infty,$$

show that Gauss-Seidel converges if the matrix A is row-diagonally dominant, using the fact that $\rho(C) \leq \|C\|_\infty$ for any matrix C.

5. Consider a **symmetric** matrix whose graph is given by:



Ties will be broken by selecting the node with the numerically earlier original label (as given on the graph) first.

[5] (a) Perform Cuthill-McKee (starting with node 1), and reverse Cuthill-McKee (starting with node 1) re-orderings on this graph.

[5] (b) Perform minimum degree re-ordering on this graph.

6. In PDE-based **image denoising**, using a simple Laplacian/diffusion approach produces overly-smoothed images that fail to preserve important edges in the data. Approaches based on "total variation regularization" tend to fare much better. This can be expressed by solving the following equation, repeatedly (for $k \geq 0$):

$$-\alpha \nabla \cdot \tfrac{1}{|\nabla u^k|} \nabla u^{k+1}(x,y) + u^{k+1}(x,y) \;=\; u^0(x,y) \qquad \text{in } \Omega = (0,1) \times (0,1), \tag{4}$$

where we will use

$$\begin{aligned} \nabla u &= \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y} \right), \text{ and} \\ |\nabla u| &= \sqrt{\left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial u}{\partial y} \right)^2}. \end{aligned}$$

The parameter $\alpha > 0$ controls the degree of smoothing to be applied. Figure 1 shows an example of an input image, the same image corrupted with noise, and a denoised image using the total variation approach.

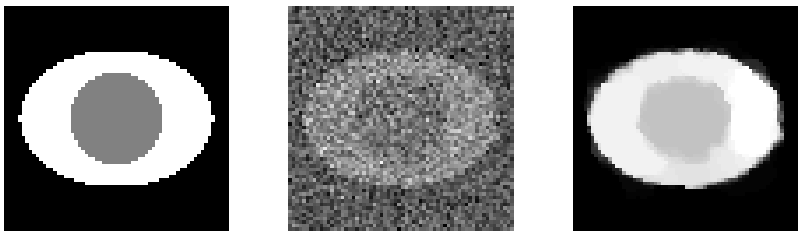


Figure 1: (Left) original, (middle) noisy, and (right) denoised images.

One possible finite difference discretization of the PDE described above gives the following equation at each grid point (x_i, y_j):

$$AC u_{i,j}^{k+1} + AW u_{i-1,j}^{k+1} + AE u_{i+1,j}^{k+1} + AS u_{i,j-1}^{k+1} + AN u_{i,j+1}^{k+1} = u_{i,j}^0, \tag{5}$$

where

$$
\begin{aligned}
AW &= -\frac{\alpha}{h^2}\left(\frac{1}{2\sqrt{\left(\frac{u^k_{i,j}-u^k_{i-1,j}}{h}\right)^2+\left(\frac{u^k_{i,j}-u^k_{i,j-1}}{h}\right)^2+\beta}}+\frac{1}{2\sqrt{\left(\frac{u^k_{i,j}-u^k_{i-1,j}}{h}\right)^2+\left(\frac{u^k_{i-1,j+1}-u^k_{i-1,j}}{h}\right)^2+\beta}}\right)\\
AE &= -\frac{\alpha}{h^2}\left(\frac{1}{2\sqrt{\left(\frac{u^k_{i+1,j}-u^k_{i,j}}{h}\right)^2+\left(\frac{u^k_{i+1,j}-u^k_{i+1,j-1}}{h}\right)^2+\beta}}+\frac{1}{2\sqrt{\left(\frac{u^k_{i+1,j}-u^k_{i,j}}{h}\right)^2+\left(\frac{u^k_{i,j+1}-u^k_{i,j}}{h}\right)^2+\beta}}\right)\\
AS &= -\frac{\alpha}{h^2}\left(\frac{1}{2\sqrt{\left(\frac{u^k_{i,j}-u^k_{i-1,j}}{h}\right)^2+\left(\frac{u^k_{i,j}-u^k_{i,j-1}}{h}\right)^2+\beta}}+\frac{1}{2\sqrt{\left(\frac{u^k_{i+1,j-1}-u^k_{i,j-1}}{h}\right)^2+\left(\frac{u^k_{i,j}-u^k_{i,j-1}}{h}\right)^2+\beta}}\right)\\
AN &= -\frac{\alpha}{h^2}\left(\frac{1}{2\sqrt{\left(\frac{u^k_{i+1,j}-u^k_{i,j}}{h}\right)^2+\left(\frac{u^k_{i,j+1}-u^k_{i,j}}{h}\right)^2+\beta}}+\frac{1}{2\sqrt{\left(\frac{u^k_{i,j+1}-u^k_{i-1,j+1}}{h}\right)^2+\left(\frac{u^k_{i,j+1}-u^k_{i,j}}{h}\right)^2+\beta}}\right)\\
AC &= -(AW+AE+AS+AN)+1.
\end{aligned}
$$

Note that the coefficients above depend on the data at step k rather than $k+1$. Here h is grid cell size (width), and $\beta = 10^{-6}$ is a constant parameter to circumvent numerical issues (division by zero). The finite difference equations can be formulated into a linear system of the form

$$A(u^k)u^{k+1} = u^0, \tag{6}$$

where the coefficient matrix $A(u^k)$ depends on the current values of u^k. The matrix's sparsity pattern of nonzeros is the same as the standard 5-point stencil 2D Laplacian matrix. You can assume zero boundary conditions.

Our basic algorithm for image denoising process will be:

Given a noisy image u^0.
for $k = 0, 1, \ldots, K$
    Solve $A(u^k)\, u^{k+1} = u^0$, for u^{k+1}
end

where K is some specified number of iterations of denoising to apply.

To actually solve equation (6) inside the loop, we will apply several of the iterative methods we have seen in class. Let $u^{k+1,\ell}$ be the approximate solution of u^{k+1} given by ℓ iterations of an iterative method. Then the denoising algorithm can be written as:

Given noisy image u^0.
for $k = 0, 1, \ldots, K$
    $u^{k+1,0} = u^k$
    for $\ell = 0, 1, \ldots$, until convergence
        $u^{k+1,\ell+1} = \text{IterativeMethodStep}(u^{k+1,\ell}, A(u^k), u^0)$
    end
end

For this question, we will use $K = 8$. For α, we will use $\alpha = 6.4 \times 10^{-2}, 3.2 \times 10^{-2}$, 1.6×10^{-2}, and 8×10^{-3} for image sizes of $16 \times 16, 32 \times 32$, 64×64, and 128×128, respectively.

[10] (a) Create two MATLAB functions:

```
A = FormMatrix(u, alpha)
u0 = FormRHS(X)
```

The input for FormMatrix is an approximate solution vector u (corresponding to the u^k terms in the coefficients for equation (4)) and the output is the matrix corresponding to equation (4). Be sure to make use of Matlab's sparse matrix functionality! The input for FormRHS is a noisy image X and the output is the vector representation u0 of X. Note that if the image size of X is $m \times m$, then the size of A should be $n \times n$ where $n = m^2$ and the size of u0 is $n \times 1$. Noisy images of different sizes can be generated by the provided Matlab file set_image, which takes an integer specifying the resolution along one dimension. (You may assume the image is always square.)

[12] (b) Implement the following iterative methods: Jacobi, Gauss-Seidel, SOR, and Conjugate Gradient, by creating the following MATLAB functions:

```
[x, iter] = Jacobi(A, b, x_initial, maxiter, tol)
[x, iter] = GS(A, b, x_initial, maxiter, tol)
[x, iter] = SOR(omega, A, b, x_initial, maxiter, tol)
[x, iter] = CG(A, b, x_initial, maxiter, tol)
```

These functions take as inputs the sparse matrix A, the right-hand side b, the initial guess x_initial , the maximum number of iterations maxiter, and the tolerance tol, and computes the approximate solution x using iter steps of the corresponding iterative method. (SOR has one more input parameter, omega.) You may use MATLAB's built-in backslash operator ('\') to perform the necessary triangular solves in the inner loops of Gauss-Seidel and SOR. For Jacobi, you may explicitly construct and use D^{-1} (i.e., breaking our usual rule of not inverting matrices).

The solvers should be stopped either when they reach the maximum number of solver iterations or when the residual vector satisfies:

$$\|r\|_2 \leq tol\, \|b\|_2.$$

[18] (c) Create a MATLAB script, Denoise, that solves the system (5) to perform denoising, using different iterative methods. In particular, for each iteration k, set up the matrix A and right-hand side $b = u0$ by calling FormMatrix and FormRHS. Then solve the linear system by calling one of the iterative methods in part (b). (Tip: for easier debugging, you may want to temporarily use MATLAB's backslash operator, A\b, for the solve first to make sure your outer loop and the two functions, FormMatrix and FormRHS, are correct, i.e., get the overall denoising working before implementing your own linear solvers. Then, replace A\b with the various iterative methods.)

In this assignment, use a relatively large solver tolerance, $tol = 10^{-2}$. Impose a maximum of 20000 solver iterations (note: this is separate from the outer/denoising iterations). For SOR, determine the optimal *omega* (requiring the fewest *total* iterations for the whole denoising process) for each grid size up to 2 decimal places by trial-and-error (by hand or by code) for the three image sizes specified below. Report the optimal values you found.

Record the CPU times and number of iterations. Construct a table of execution times and a table of *total* number of iterations for the denoising process when using Jacobi, Gauss-Seidel, SOR (with your optimal omega values), and Conjugate Gradient iterations. Use image sizes 16×16, 32×32, and 64×64 (and *optionally*, 128×128, if you would like and time permits). Comment on the results. Convert the output vector back to a matrix (2D image), and use **imagesc** with a grayscale color map to display the denoised image.

Submit:

- Your code for FormMatrix, FormRHS, Jacobi, GS, SOR, CG and Denoise;
- Your list of optimal omega values.
- The plots of the denoised images for the grid sizes specified above;
- Your table of CPU times and iteration counts;
- Comments on your observations.

Once again, submit a copy of everything to CrowdMark in PDF/image format, and separately submit your [runnable] source code to the A03 LEARN DropBox in a single ZIP file.