CS 475/CM 375 - Fall 2010: Assignment 2

Instructor: Pascal Poupart   Office: DC2514   Email: ppoupart@uwaterloo.ca
Classroom: MC 4060   TuTh 10:00am-11:20pm   Office hours: TuTh 2:30-4pm
Class homepage: www.student.cs.uwaterloo.ca/~cs475/

Due: November 2, Tuesday (in class)

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

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

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

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

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

2. (15 marks) Let A be a strictly **row** diagonally dominant matrix; i.e.

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

 (a) Show that Jacobi method converges.
    (Hint: you could use the result that $\rho(J) \leq \|J\|_\infty$ for any matrix J. Recall the definition of infinity norm:

$$\|x\|_\infty \;\; = \;\; \max_i |x_i|,$$

$$\|J\|_\infty \;\; = \;\; \max_i \sum_{j=1}^n |J_{ij}|.$$

    Consider J = iteration matrix of the Jacobi method.)

 (b) Let G be the iteration matrix of the Gauss-Seidel method; i.e.

$$G = I - (D - L)^{-1} A$$

    where $D - L$ is the lower triangular part of A. Suppose x is any vector with $\|x\|_\infty = 1$, and let $y = Gx$. Show that $\|y\|_\infty < 1$. (Hint: consider $(D - L)y$.)

(c) By definition of matrix norm,

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

Use part (b), show that the Gauss-Seidel method converges.

3. (10 marks) Let $\{p^i\}$ be a set of A-orthogonal search direction vectors. We want to look for x^{k+1} in *all* of these directions. Thus, we write

$$x^{k+1} = x^0 + \sum_{i=0}^{k} \alpha_i p^i.$$

We determine $\{\alpha_i\}$ by minimizing $F(x^{k+1})$, where

$$F(x) \equiv \frac{1}{2} x^T A x - b^T x = \frac{1}{2}(x, x)_A - (b, x),$$

over all search directions.

(a) Show that

$$F(x^{k+1}) = \frac{1}{2}(x^0, x^0)_A + \sum_{i=0}^{k} \alpha_i (x^0, p^i)_A + \frac{1}{2} \sum_{i=0}^{k} \sum_{j=0}^{k} \alpha_i \alpha_j (p^i, p^j)_A - (b, x^0) - \sum_{i=0}^{k} \alpha_i (b, p^i).$$

(b) By using the A-orthogonal property, show that

$$F(x^{k+1}) = \frac{1}{2}(x^0, x^0)_A + \sum_{i=0}^{k} \alpha_i (x^0, p^i)_A + \frac{1}{2} \sum_{i=0}^{k} \alpha_i^2 (p^i, p^i)_A - (b, x^0) - \sum_{i=0}^{k} \alpha_i (b, p^i).$$

(c) To minimize $F(x^{k+1})$, we set $\frac{\partial F}{\partial \alpha_j}(x^{k+1}) = 0$. Show that

$$\alpha_j = \frac{(r^0, p^j)}{(p^j, p^j)_A}.$$

Thus α_j depends only on p^j, not on any other search directions. Once we have minimized in direction p^j, we are done with that direction. In other words, each of the p^j minimizes $F(x^{k+1})$ in a subspace and we *never* have to look in that subspace again.

4. (15 marks) Consider the least squares problem $Ax = b$ where

$$A = \begin{bmatrix} 3 & -3 & -5 \\ 0 & 4 & 8 \\ 4 & -4 & -15 \\ 0 & 3 & 6 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

(a) Solve the least squares problem using normal equations.

(b) Solve the least squares problem using (classical or modified) Gram-Schmidt. Determine the $\hat{Q}$ and $\hat{R}$ factors.

2

(c) Solve the least squares problem using Householder transformation. Determine the Q and R factors.

5. (40 marks) In PDE-based image processing, the simple diffusion approach produces images with smeared edges. Employing the total variation regularization approach, images with sharp edges can be restored by solving the following equation repeatedly (for $k \geq 0$):

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

where $\nabla u = (\partial u / \partial x, \partial u / \partial y)$ and $|\nabla u| = \sqrt{(\partial u / \partial x)^2 + (\partial u / \partial y)^2}$. $\alpha > 0$ is the parameter controlling how much noise is to be removed. Figure 1 shows a reconstructed image after 20 iterations.
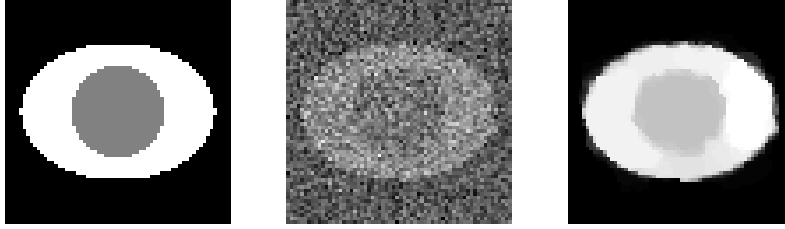

Figure 1: (Left) original, (middle) noisy, and (right) denoised images.

Applying a standard finite difference discretization, at each grid point (x_i, y_j), we have

$$AC u_{i,j}^{k+1} + AW u_{i-1,j}^{k+1} + AE u_{i+1,j}^{k+1} + AS u_{i,j-1}^{k+1} + AN u_{i,j+1}^{k+1} = u_{i,j}^0, \qquad (2)$$

where

$$AW = -\frac{\alpha}{h^2} \left(\frac{1}{2\sqrt{(\frac{u_{i,j}^k - u_{i-1,j}^k}{h})^2 + (\frac{u_{i,j}^k - u_{i,j-1}^k}{h})^2 + \beta}} + \frac{1}{2\sqrt{(\frac{u_{i,j}^k - u_{i-1,j}^k}{h})^2 + (\frac{u_{i-1,j+1}^k - u_{i-1,j}^k}{h})^2 + \beta}} \right)$$

$$AE = -\frac{\alpha}{h^2} \left(\frac{1}{2\sqrt{(\frac{u_{i+1,j}^k - u_{i,j}^k}{h})^2 + (\frac{u_{i+1,j}^k - u_{i+1,j-1}^k}{h})^2 + \beta}} + \frac{1}{2\sqrt{(\frac{u_{i+1,j}^k - u_{i,j}^k}{h})^2 + (\frac{u_{i,j+1}^k - u_{i,j}^k}{h})^2 + \beta}} \right)$$

$$AS = -\frac{\alpha}{h^2} \left(\frac{1}{2\sqrt{(\frac{u_{i,j}^k - u_{i-1,j}^k}{h})^2 + (\frac{u_{i,j}^k - u_{i,j-1}^k}{h})^2 + \beta}} + \frac{1}{2\sqrt{(\frac{u_{i+1,j-1}^k - u_{i,j-1}^k}{h})^2 + (\frac{u_{i,j}^k - u_{i,j-1}^k}{h})^2 + \beta}} \right)$$

$$AN = -\frac{\alpha}{h^2} \left(\frac{1}{2\sqrt{(\frac{u_{i+1,j}^k - u_{i,j}^k}{h})^2 + (\frac{u_{i,j+1}^k - u_{i,j}^k}{h})^2 + \beta}} + \frac{1}{2\sqrt{(\frac{u_{i,j+1}^k - u_{i-1,j+1}^k}{h})^2 + (\frac{u_{i,j+1}^k - u_{i,j}^k}{h})^2 + \beta}} \right)$$

$$AC = -(AW + AE + AS + AN) + 1.$$

Here h is mesh size, and $\beta = 10^{-6}$ is a constant parameter. Conceptually, the finite difference equation (2) corresponds to solving the linear system:

$$A(u^k)u^{k+1} = u^0, \qquad (3)$$

3

where the coefficient matrix $A(u^k)$ depends on the current values of u^k and its nonzero structure is the same as the standard 5-point stencil 2D Laplacian matrix. (Assume zero boundary condition.)

To summarize, the algorithm of the image denoising process is:

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

To solve equation (3), we apply different iterative methods. Let $u^{k+1,j}$ be the approximate solution of u^{k+1} given by j 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 $j = 0, 1, \ldots$, until convergence
> $u^{k+1,j+1} = \text{IterativeMethod}(u^{k+1,j}, A(u^k), u^0)$
>   end
> end

(For this assignment, use $K = 10$, and $\alpha = 4 \times 10^{-2}, 3 \times 10^{-2}, 1.5 \times 10^{-2}$, and 1.2×10^{-2} for the image size $16 \times 16, 32 \times 32, 64 \times 64$, and 128×128, respectively.)

(a) Create a MATLAB function:

$$[\text{A},\text{u0}] \; = \; \text{ImageMatrix(X)}$$

The input is the noisy image X and the outputs are the matrix A (equation (2)) and 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 `set_image` which can be downloaded from the class homepage. Submit a listing of your code.

(b) Implement the iterative methods: Jacobi, Gauss-Seidel, SOR and Conjugate Gradient. Create the following MATLAB functions:

$$\text{x} = \text{Jacobi(A,b,x\_initial,maxiter,tol)}$$
$$\text{x} = \text{GS(A,b,x\_initial,maxiter,tol)}$$
$$\text{x} = \text{SOR(omega,A,b,x\_initial,maxiter,tol)}$$
$$\text{x} = \text{CG(A,b,x\_initial,maxiter,tol)}$$

These MATLAB functions take as inputs the matrix A, the right-hand size b, the initial guess $x\_initial$, the maximum number of iterations $maxiter$ and the tolerance tol, and compute the approximate solution x using the corresponding iterative method. (SOR has one more

4

input parameter, $omega$.) The iteration stops either when it reaches the maximum number of iterations or the residual vector satisfies:

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

Note that for Jacobi, Gauss-Seidel, and SOR, r^j is not readily available. Since Jacobi and Gauss-Seidel usually take thousands of iterations to converge, you may want to do the convergence test every 100 iterations to save up time. For SOR, do the convergence test every 10 iterations.

Assume that A is sparse and it has the 5-point stencil nonzero structure. Thus, A has at most 5 nonzeros per row. You can and you should hard code this assumption in your implementations for optimal speed. Also, for GS and SOR, keep only one vector. Submit all your code.

(c) Create a MATLAB script, `Denoise`, that solves (3) using different iterative methods. In particular, for each k, set up the matrix A and right-hand side $b = u0$ by calling `ImageMatrix`. Then solve the linear system by calling one of the iterative methods in part (b). (Note: if you have trouble setting up correctly the matrix in part (a), you may replace A by `Lap2D` in assignment 1 for partial credits.)

In this assignment, use a relatively large tolerance, $tol = 10^{-2}$. In practice, one usually uses a much smaller tolerance, e.g. 10^{-6}. However, for image denoising, the result is not distinguishable by eyes; more importantly, the number of iterations can be dramatically reduced. Also, impose the maximum number of iterations, say, 100000. For SOR, determine the optimal $omega$ to 2 significant digits by trial-and-error and report the value.

Record the CPU times and number of iterations. Construct a table of execution times and a table of total number of iterations for solving (2) using Jacobi, Gauss-Seidel, SOR and Conjugate Gradient iterations. Use image size 16×16, 32×32, 64×64, 128×128 (optional, depending on time). Comment on the results. Use `x1to2.m` and the MATLAB command `imshow` or `imagesc` to display the denoised image. Submit the denoised images for the grid sizes specified above.