

University of Delaware, Department of Mathematical Sciences
 Math 353 Engineering Mathematics III S08, Instructor: Constantin Bacuta
 Project I (Maximum = 40 points)

The goal of this project is to learn how to program Jacobi and Gauss-Seidel algorithms, to do numerical experiments and to solve practical problems with MATLAB build in functions and with functions written by you.

For Part I and Part II: Consider the following tridiagonal system, and assume that the coefficient matrix is strictly diagonally dominant

$$\begin{cases} d_1 x_1 + c_1 x_2 & = b_1 \\ a_{k-1} x_{k-1} + d_k x_k + c_k x_{k+1} & = b_k, \text{ for } k = 2, 3, \dots, n-1, \\ a_{n-1} x_{n-1} + d_n x_n & = b_n \end{cases} \quad (1)$$

Part I

- (i) Write a Jacobi algorithm and a Gauss-Seidel algorithm to solve a general tri-diagonal system. Your algorithms should *efficiently use the “sparseness” of the coefficient matrix*.
- (ii) Write two MATLAB functions called **jac3diag** and **gs3diag** for the corresponding two algorithms from part (i). For both functions the input should be the **three vectors** a, d, c which define the tri-diagonal matrix, the initial guess x_0 as a $n \times 1$ vector, the $n \times 1$ vector b which defines the right hand side of the system, the tolerance δ (take $\delta = 1e - 12$), and $Nmax$ the maximum number of iterations (take $Nmax = 100$). As output, consider the $n \times 1$ vector x which “solves” the system, and the number of iterations k performed by the algorithm. The algorithms should stop when $err := \|x^{(k+1)} - x^{(k)}\|$ is smaller than δ or, when the maximum number of iterations is achieved. Here, $x^{(k)}$ and $x^{(k+1)}$ are two consecutive iterations and $\|x\| := \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$ for $x = (x_1, x_2, \dots, x_n)^T \in R^n$.
- (iii) Debug your programs, by using the following example:

```
>>a=1:4
>>d=8*ones(1,5)
>>c=-1:-1:-4
>>A=diag(a,-1) +diag(d,0) +diag(c,1)
>>b=[7 7 7 7 12]’
Solve the system Ax=b with >>x=A\b
```

Solve the above system $Ax = b$ using your **jac3diag** or **gs3diag**. The answer should be the same. **If not, go back and fix your program(s).**

- (iv) Solve the system (1) for $a = ones(1, n - 1)$, $d = 4 * ones(1, n)$, $c = ones(1, n - 1)$, $b = 3 * ones(n, 1)$, and $n = 50$, using your **jac3diag** and **gs3diag**. Check the numerical solution by using the the function **testproj1.m** bellow.

```

function testproj1(n)
tic
a=ones(1, n-1); d=4*ones(1,n); c=ones(1, n-1);
b=3*ones(n,1); %Creates the RHS of the system (a)
A=diag(a,-1) +diag(d,0) +diag(c,1); %Builds the tridiagonal matrix A of (a)
xGE=A\b % Solve Ax=b by using the Gaussian Elimination.
error=norm(b-A*xGE) %verifies if xGE is a good approximation to x=inv(A)*b.
toc

```

After you copy **testproj1.m** in your working directory, to solve (a), just type `>> testproj1(50)`.

- (v) The commands **tic** and **toc** in **testproj1.m** measure the elapsed time needed to solve a tri-diagonal system $Ax = b$ using the MATLAB **-backslash** (or the *left matrix divide* based on the Gaussian elimination algorithm). Use the commands **tic** and **toc** to test the elapsed time needed to solve the same tri-diagonal system with $a = \text{ones}(1, n - 1)$, $d = 4 * \text{ones}(1, n)$, $c = \text{ones}(1, n - 1)$ and $b = 3 * \text{ones}(n, 1)$ by using **jac3diag** and **gs3diag**. Run numerical tests to compare the time needed by **jac3diag**, **gs3diag** and the **backslash**-Gaussian elimination to solve the system for $n = 100, 200, 1000, 2000$. For testing the time for **jac3diag** and **gs3diag** you might create *function M-files* similar with the **testproj1.m** function used for the **backslash** solver. For what value n does the **backslash** fails to work? (Use **testproj1.m** to answer this). Does **jac3diag** or **gs3diag** work for that value ? Comment on the results.

For Part I (ii)-(iv), attach the script files and the function M-files created by you. Provide the MATLAB commands used for each item. Comment on the results.

Part II: Practical Application. The system $Ax = b$ considered in (1), with $A = \text{diag}(a, -1) + \text{diag}(d, 0) + \text{diag}(c, 1)$, where $a = \text{ones}(1, n - 1)$, $d = -2 * \text{ones}(1, n)$, $c = \text{ones}(1, n - 1)$ describes the following application. An elastic band of length 1 is pinned down at the ends $t = 0$ and $t = 1$. At the n intermediate points $t_1 = 1/(n + 1)$, $t_2 = 2/(n + 1)$, \dots , $t_n = n/(n + 1)$, the weights b_1, b_2, \dots, b_n are suspended. This causes the elastic band to deflect, with the deflection at the the point $t_k = k/(n + 1)$ proportional to x_k . The deflection vector $x = [x_1, x_2, \dots, x_n]^T$ can be determined as the exact solution of the system $Ax = b$ with $b = [b_1, b_2, \dots, b_n]^T$.

- (i) Take $n = 20$ and use **backslash** or **trisys** (from the class home page) to find the deflection x for constant weights b_k ($b = \text{ones}(n, 1)$). Plot the deflection vector against the nodes t_k . To show the two end points where the deflection is zero, an efficient plotting command is `>> plot(linspace(0, 1, n + 2), [0; x(:); 0]')` where x is the column vector which solves $Ax = b$. Repeat the experiment with $n = 50$ and $n = 100$.
- (ii) Take $n = 200$. Put unit weights on the left half of the band and zero weights on the right half ($b = [\text{ones}(100, 1); \text{zeros}(100, 1)]$) and determine the resulting deflection.
- (iii) Take $n = 101$. Put equal weights on the middle 21 nodes, of the band (take e.g., $b_{51} = 1/100$), and zero weights on the rest of the nodes. Determine the resulting deflection.
- (iv) Investigate other weights. Can you use your **jac3diag** or **gs3diag** to solve this problem?

For Part II provide the plots, the commands used to get the plots, and meaningful comments.

Extra-Credit = 10 Points. Write a function M-file to solve a general tridiagonal system (1) using SOR method. Your algorithms should *efficiently use the “sparseness” of the coefficient matrix* and have ω as one of the input arguments. Use your “mySOR.m” to solve the system of Problem I-(iv). Find values ω for which SOR performs better (fewer iterations to achieve a fixed tolerance say, $\epsilon = 0.5 * 1e - 9$) than Jacobi or GS iterations.

Note: The project should be well written and well commented. **The length of the project should not exceed 4 pages**, not counting the plots, the MATLAB files created by you, and the extra credit part. Neatness counts. *A (good) picture is worth a thousand words.*

A (good) comment is worth a thousand lines of MATLAB output. Save the trees.