Norwegian University of Science
and Technology
Department of Mathematical
Sciences

MA2501 Numerical Methods
Spring 2017

Exercise set Semester Project

Practical Information

The deadline for the project is **April 27 2017, at 24:00**.

- This project counts for 30% of the final grade.
- You have to work in groups.
- You have to produce a report written on a computer with your solutions (preferably in LaTeX). You do not need to include codes in your report. It is enough to refer to the produced code files. It could be helpful for you to go through the tasks point by point and answer them. Just try to keep the document organized and readable.
- Try to write efficient, organized, and well documented MATLAB code.
- The report and the codes should be sent electronically to Abdullah Abdulhaque via email (abdullah.abdulhaque@ntnu.no). Mark all the material with your candidate number(s), not your name(s).

MATLAB advice

All the MATLAB codes should satisfy the following criterions:

- Function files should contain a help text. A user should be able to use the routine from the information given by the command:
  `help [function name]`.
- The code should be self-documented, with a reasonable amount of comments in the codes.

Try to make use of MATLAB's proficiency at working with vectors and matrices, i.e. avoid unnecessary for-loops. Avoid repeating the same computation several times rather than computing the result once and storing it.

Exercise 1

Theoretical background

Consider Poisson's equation on a rectangle with homogeneous Dirichlet conditions:

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f(x,y) \quad , \quad (x,y) \in \Omega = (0, L_x) \times (0, L_y) \tag{1a}$$

$$u = 0 \quad , \quad (x,y) \in \partial\Omega \tag{1b}$$

The second-order derivatives can be discretized with the central difference scheme:

$$\frac{u(x_i - h, y_j) - 2u(x_i, y_j) + u(x_i + h, y_j)}{h_x^2} = \left(\frac{\partial^2 u}{\partial x^2}\right)\Big|_{(x_i, y_j)} + \mathcal{O}(h_x^2) \tag{2a}$$

$$\frac{u(x_i, y_j - h) - 2u(x_i, y_j) + u(x_i, y_j + h)}{h_y^2} = \left(\frac{\partial^2 u}{\partial y^2}\right)\Big|_{(x_i, y_j)} + \mathcal{O}(h_y^2) \tag{2b}$$

In this construction, $h_x = L_x/N_x$ and $h_y = L_y/N_y$ are the *uniform mesh widths*, N_x and N_y are the *number of subintervals*, and $0 \le i \le N_x$ and $0 \le j \le N_y$. If $\Omega = (0,1)^2$, the unit square, and $N_x = N_y = N$, we obtain the *5-point stencil* for equation (1):

$$-u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1} + 4u_{i,j} = h^2 f_{i,j} \tag{3}$$

In this way, we get a linear system of equations $\mathbf{Au} = \mathbf{f}$, where

$$\mathbf{A} = \begin{bmatrix} \mathbf{S} & \mathbf{T} & & & \\ \mathbf{T} & \mathbf{S} & \mathbf{T} & & \\ & \ddots & \ddots & \ddots & \\ & & \mathbf{T} & \mathbf{S} & \mathbf{T} \\ & & & \mathbf{T} & \mathbf{S} \end{bmatrix} \quad , \quad \mathbf{f} = h^2 \begin{bmatrix} f_{1,1} \\ f_{1,2} \\ \vdots \\ f_{M,M-1} \\ f_{M,M} \end{bmatrix}$$

$$\mathbf{T} = \begin{bmatrix} -1 & & & & \\ & -1 & & & \\ & & \ddots & & \\ & & & -1 & \\ & & & & -1 \end{bmatrix} \quad , \quad \mathbf{S} = \begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{bmatrix}$$

The original size of $\mathbf{S}$ and $\mathbf{T}$ is $(N+1) \times (N+1)$, but because of the boundary conditions, the size reduces to $M \times M$, where $M \equiv N - 1$.

In MATLAB, it is possible to define the matrices with the commands

```
e = ones(M,1)  ,  S = full(spdiags([-e 4*e -e],-1:1,M,M))  ,  T = diag(-e)
```

The vector $\mathbf{f}$ can be defined as a matrix where $f_{ij} = h^2 f(x_i, y_j)$, and then we can transform it to a vector with the command `f = f(:)`.

Exercise 1 a)

Implement matrix **A** and vector **f** in MATLAB, and let $f(x, y) = 8\pi^2 \sin(2\pi x) \sin(2\pi y)$ be the source function at the right-hand side of (1).

Exercise 1 b)

Implement the following algorithms in MATLAB:

- Gaussian elimination (GE)
- Jacobi iteration (J)
- Gauss-Seidel iteration (GS)
- Richardson iteration (R)
- Successive overrelaxation iteration (SOR)

The MATLAB-file for Gaussian elimination just needs to return the **numerical solution** and the **total elapsed running time**. To accelerate the speed, you can utilize the sparsity pattern of the matrix to reduce the number of for-loops.
Before starting the MATLAB-files for iteration, you can make the matrix sparse. These files must return the **numerical solution**, the **number of iterations**, the **residual error**, and the **total elapsed running time**. The iterative algorithms should stop after reaching a certain error tolerance. Define this tolerance as 10^{-14} such that you obtain machine precision. The residual error is calculated at the end and reveals whether the code really worked. It is defined as

$$r^{(N)} = \frac{\|b - Ax^{(N)}\|_2}{\|b\|_2} \tag{4}$$

where N is the final number of iterations used. Choose the initial guess as `ones(n,1)/n`.
Hint: Test these methods on small linear systems to verify that they work correctly first.
Useful MATLAB-functions: `norm`, `sparse`, `tril`, `eye`, `diag`, `tic`, `toc`

Exercise 1 c)

Write a MATLAB-program which executes the different algorithms (not SOR) for different values of N. Let `N = 9:10:79`. Extract the number of iterations, total running time and residual error from the function output and store them in an .asc-file (for Gaussian elimination, just return the running time).
Useful MATLAB-functions: `fopen`, `fclose`, `fprintf`

Exercise 1 d)

Write a MATLAB-program which opens the .asc-file, extracts all the data, stores them in appropriate vectors, and plots the convergence graphs for each method. There should be one figure for the number of iterations, one for the time, and one for the residual error. Make the graphs logarithmic. The x-axis displays the size of the matrices. Analyze the graphs and comment the methods' performance.
Useful MATLAB-functions: `fscanf`, `strcat`, `num2str`, `reshape`, `figure`, `hold on`, `loglog`, `xlabel`, `ylabel`, `legend`, `saveas`, `gcf`, `end`

Exercise 1 e)

Let ω `= 0.2:0.2:1.8`, `N = 3:4:19`, and test the performance of the SOR iteration. This time, the x-axis displays the acceleration parameter. How does the number of iterations and total running time vary as a function of ω and N? Which value of ω seems to yield the best result?
Hint: You can reuse much of the code for simulation and plotting.

Exercise 1 f)

Let `N = 7:8:79`, and plot the spectral radius of $\mathbf{A}$ and the iteration matrices (J, GE, R) as functions of N. Comment the behaviour of the graphs. How does the spectral radius vary with N? Do you see any pattern?
Let ω `= 0.2:0.2:1.8`, `N = 7:8:39`, and repeat the same process for the SOR-method. Comment the results. Which algorithm seems to have performed best for this exercise?
Useful MATLAB-functions: `sparse`, `eig`, `abs`, `max`

Exercise 2

Theoretical background

Consider Helmholtz's equation on a rectangle with homogeneous Dirichlet conditions:

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} + \mu u = f(x,y) \quad , \quad (x,y) \in \Omega = (0, L_x) \times (0, L_y) \tag{5a}$$

$$u = 0 \quad , \quad (x,y) \in \partial\Omega \tag{5b}$$

where $\mu > 0$. If we use the finite difference method again for discretizing this PDE, then the *modified 5-point stencil* becomes

$$-u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1} + (4+\mu)u_{i,j} = h^2 f_{i,j} \tag{6}$$

Exercise 2 a)

Implement matrix $\mathbf{A}$ and vector $\mathbf{f}$ in MATLAB as you did previously. Let $\mu = 4\pi^2$ and let $f(x, y) = 12\pi^2 \sin(2\pi x) \sin(2\pi y)$ be the source function at the right-hand side of (5).

Exercise 2 b)

Try the different algorithms (not SOR) for different values of `N = 9:10:79`. Extract the number of iterations, the total running time, and the residual error from the output.

Exercise 2 c)

Plot the number of iterations and running time for each method, in the same way as earlier. Analyze the graphs and comment the methods' performance.

Exercise 2 d)

Let ω `= 0.2:0.2:1.8`, `N = 3:4:19`, and test the performance of the SOR iteration. How does the number of iterations and total running time vary as a function of ω and N? Which value of ω seems to yield the best result?

Exercise 2 e)

Let `N = 7:8:79`, and plot the spectral radius of $\mathbf{A}$ and the iteration matrices as functions of N. Comment the behaviour of the graphs. How does the spectral radius vary with N? Do you see any pattern?
Let ω `= 0.2:0.2:1.8`, `N = 7:8:39`, and repeat the same process for the SOR-method. Comment the results. Which algorithm seems to have performed best for this exercise?

Exercise 3

Theoretical background

The Hilbert matrix **H** is a well-known matrix with very bad condition number. The entries of this matrix is given by the formula:

$$h_{ij} = \frac{1}{i+j-1} \tag{7}$$

We will perform the previous analysis on this new matrix in this exercise. Since the matrix is full, there is no need for making it sparse. Choose the right-hand side as `ones(n,1)`.

Exercise 3 a)

Implement the Hilbert matrix in MATLAB. Try the different algorithms (not SOR) for different values of `N = 3:6`. Extract the number of iterations, the total running time, and the residual error from the function output and store them.

Exercise 3 b)

Plot the number of iterations and running time for each method, in the same way as earlier. Analyze the graphs and comment the methods' performance.

Exercise 3 c)

Let ω `= 0.2:0.2:1.8`, `N = 3:4`, and test the performance of the SOR iteration. How does the number of iterations and total running time vary as a function of ω and the size of the system? Which value of ω seems to be the best?

Exercise 3 d)

Let `N = 3:40`, and plot the spectral radius of **H** and the iteration matrices as functions of N. Comment the behaviour of the graphs. How does the spectral radius vary with N? Do you see any pattern?
Let ω `= 0.2:0.2:1.8`, `N = 8:8:40`, and repeat the same process for the SOR-method. Comment the results. Which algorithm seems to have performed best for this exercise?