FMN011 Exam — Computational Part I
Solving huge systems of equations

Lund, March 2012

Please read the detailed instructions on the course webpage.

Hand in the report on paper (not via e-mail) during the break (11:00-11:15 a.m.) at the seminar on Thursday, April 19, 2012, or else place it in the box marked FMN011 at the bottom of the shelf located at the entrance of the right-hand side corridor (MH, ground floor). The reports will be collected at 12:10 sharp on Thursday, April 19, 2012. Any report handed in or placed in the box after this time will not be accepted.

Around 1976 I took my first course in Numerical Analysis. Our computational task was to implement the SOR method and solve a system of 9 equations. Today, more than 35 years later, you are given basically the same task, except that instead of solving a system of 7 equations you are asked to solve one with hundreds of thousands or even millions of unknowns. Linear systems of this size are common in several real-life applications. In most of them the system matrix is sparse (only $O(n)$ elements are nonzero), and special techniques for sparse matrices can and should be used.

Sometimes slightly different large sparse systems must be solved sequentially in real time, and here iterative methods prove practical because the solution from the previously solved problem can be used as the initial guess. A good starting guess will require fewer iterations to reach a prescribed accuracy. In this project you will compare different methods for solving large scale sparse linear systems repeatedly.

We will start with the matrix defined as follows. For an $n \times n$ matrix A, with n even, let

$$\begin{aligned}
A(i,i-1) &= -1, \quad \text{for} \quad i = 2,\ldots,n \\
A(i,i) &= 3, \quad \text{for} \quad i = 1,\ldots,n \\
A(i,i+1) &= -1, \quad \text{for} \quad i = 1,\ldots,n-1 \\
A(i,n+1-i) &= 0.5, \quad \text{for} \quad i = 1,\ldots,n, \quad \text{except for} \quad i = n/2, n/2+1
\end{aligned}$$

For instance, if $n = 8$,

$$A = \begin{pmatrix} 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ -1 & 3 & -1 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & -1 & 3 & -1 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & -1 & 3 & -1 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & -1 & 3 & -1 \\ 0.5 & 0 & 0 & 0 & 0 & 0 & -1 & 3 \end{pmatrix}$$

We will construct a vector b so that the solution to $Ax = b$ is $x = [1, 1, \ldots, 1, 1]^T$, that is, make $b = Ax$ for the given A and x. Thus, when you solve $Ax = b$ numerically you will be able to calculate the error of the numerical solution. For $n = 8$ we get $b = [2.5,\ 1.5,\ 1.5,\ 1,\ 1,\ 1.5,\ 1.5,\ 2.5]$. (Read section 2.5.4 in Sauer's book).

Task 1. Calculate the number of nonzero elements in an $n \times n$ matrix A with the same structure as above.

Task 2. Write a program that sets up matrix A and vector b as described above, using the MATLAB built-in functions `spdiags` and `fliplr`. The number of equations, n, should be an input parameter. Make the numbers corresponding to $\{-1, 3, -1, 0.5\}$ in the definition of A also input parameters so that you can later change the elements of the matrix.[1]

Task 3. Notice that A is strictly diagonally dominant (why?). This fact not only guarantees that the Jacobi and Gauss-Seidel (and thus SOR) methods converge, but also that Gauss elimination without pivoting can be used.

Implement the code for naive Gauss elimination with back substitution from Sauer's book and use it to solve the system for $n = 8$.

Task 4. Find out how much you can increase k, for $n = 2^k$, before you are unable to solve the problem using the naive Gauss elimination algorithm (i.e., how many equations can you solve).

Task 5. MATLAB's backslash operator is a sophisticated implementation of Gauss elimination that takes into account the matrix structure (for instance, if it is sparse, banded, triangular, symmetric, etc.).

Repeat Task 4 using the backslash operator and compare results.[2]

Task 6. Implement the SOR algorithm from Sauer's book in a function with input parameters A, b, ω, x_0, tol, and maxit. Solve for $n = 8$ and iterate until the maximum error[3]is below 10^{-4}. Use $[0,\ 0,\ \ldots\ , 0]^T$ as the initial vector. Try out different relaxation parameters, $1 \leq \omega \leq 2$, and find the optimal ω for this problem.

[1] To visualize your matrix you may convert a sparse matrix to full with the command `full`. To visualize the nonzero structure you can us `spy`.

[2] Read MATLAB's documentation on the backslash operator.

[3] The relative error at each SOR iteration step is a vector of length n and it is equal to the absolute error (why?) What norm should you use to measure the maximum error?

Task 7. Find out how large a system you can solve with your SOR implementation. Compare results.

Task 8. You will now make your SOR code more efficient. You do not want to calculate any inverse matrices, so you will rewrite your SOR algorithm accordingly. To accomplish this, note that at each iteration step a sparse lower triangular system must be solved. To solve this system use the backslash operator, which does only an optimized forward substitution.

Task 9. The next step is to perturb both the matrix and the right hand side of the system. For a matrix with structure

$$A = \begin{pmatrix} d & t & 0 & 0 & 0 & 0 & 0 & r \\ s & d & t & 0 & 0 & 0 & r & 0 \\ 0 & s & d & t & 0 & r & 0 & 0 \\ 0 & 0 & s & d & t & 0 & 0 & 0 \\ 0 & 0 & 0 & s & d & t & 0 & 0 \\ 0 & 0 & r & 0 & s & d & t & 0 \\ 0 & r & 0 & 0 & 0 & s & d & t \\ r & 0 & 0 & 0 & 0 & 0 & s & d \end{pmatrix}$$

construct perturbed values of s, d, t and r by adding to each quantity a random number between -10^{-4} and 10^{-4}. Use MATLAB's `rand`.[4]

Perturb the right hand side vector. The solution to the perturbed system will no longer be $x = [1, 1, \ldots, 1]^T$.

Task 10. Starting with the original system $Ax = b$ with $n = 8$, solve this system and the perturbed system $\tilde{A}x = \tilde{b}$ in sequence using Gauss elimination. Measure the average execution time by solving the same problem 100 times. Use MATLAB's `tic/toc`.

Task 11. Repeat Task 10 with SOR, using the last solution as the initial guess for the next problem. Compare times when solving the same systems with Gauss elimination and SOR.

Task 12. Repeat the experiment increasing k as far as you can.

Task 13. Repeat for a large value of k, but this time solve 8 slightly different systems sequentially.

Your conclusions are very important. No results should be given without a comment.

Based on Example 2.25 in Sauer, *Numerical Analysis,* Pearson, 2012.

Carmen Arévalo
Matematikcentrum, Numerisk Analys, Lunds Universitet

[4]The vector r = -1e-4 + 2e-4.*rand(5,1) will contain 5 random numbers in the interval $[-10^{-4}, 10^{-4}]$.