Division of Scientific Computing
Department of Information Technology
Uppsala University

NGSSC: Numerical Methods in Scientific Computing
January 23, 2012

Computer lab: Direct solution methods for linear systems of algebraic equations with dense and sparse matrices

Requirements: At the end of the lab session, please deliver a Matlab-generated report on the runs you performed. Give your comments and observations (included in the generated report or added by hand). Print the report on `pr1515`.

Where to run Matlab: After logging in on some of the Unix hosts, please issue the command `linuxlogin`. For some more details, see `http://www.it.uu.se/datordrift/maskinpark/linux`.

The task of this computer lab is to experience matrices with various properties and issues related to the solution of linear systems using direct solution methods.

Exercise 1 (Feel the sparse-dense difference)
Create a 100×100 random matrix that has about 5% nonzero entries:

```
N=100; S = sprand(N,N,.05);
```

Convert it to full matrix

```
F = full(S);
```

1. Check the time required for MATLAB to compute a product of two dense and two sparse matrices:

   ```
   tic,S*S;toc
   tic,F*F;toc
   ```

   What do you see? Is the time to multiply two dense matrices larger than when we multiply the two sparse matrices?

2. Repeat the experiment for N equal to 500, 1000 and 5000. Is there a change in the proportions of the time spent for the corresponding dense and sparse matrices?

   Explain what you see. Observe first what is the computational complexity to multiply two dense, respectively, sparse matrices and check whether the results you obtain correspond to the theoretical prediction.

Exercise 2 (Ill-conditioning)
Consider the problem $A\mathbf{x} = \mathbf{b}$, where

$$A\mathbf{x} = \begin{bmatrix} 0.835 & 0.667 \\ 0.333 & 0.266 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \mathbf{b}. \tag{1}$$

Let the right-hand vector $\mathbf{b}$ be a result of an experiment and is read from the dial of a test instrument as $b_0 = \begin{bmatrix} 0.168 \\ 0.067 \end{bmatrix}$. However, due to the small uncertainty, we have to expect that

$$0.167 \leq b_1 \leq 0.169 \quad \text{and} \quad 0.066 \leq b_2 \leq 0.068.$$

Thus, the solution for $b_1 = \begin{bmatrix} 0.167 \\ 0.068 \end{bmatrix}$ and for $b_2 = \begin{bmatrix} 0.169 \\ 0.066 \end{bmatrix}$ should be expected to be as valid as that in the first case.

1. Find the exact solution of the system for each of the above vectors $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$.
2. Is the observed instability due to some numerical procedure, the vector $\mathbf{b}$, the matrix A or a combination of these factors?
3. Try to quantify how ill-condition the problem is. How would you do this?
4. Perturbe the system as follows

   $$\begin{bmatrix} 0.835 & 0.667 \\ 0.333 & 0.266 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.1669995 \\ 0.066601 \end{bmatrix}$$

   Determine the exact solution and compare it with the exact solution corresponding to $\begin{bmatrix} 0.168 \\ 0.067 \end{bmatrix}$.

   On the basis of the results formulate a statement concerning the necessity for the solution of an ill-posed system to undergo a radical change for every perturbation of the original system.

Remark 1 *One can demonstrate the reason for a 2×2-system to be ill-conditioned. Geometrically, two equations in two unknowns define two straight lines. The point of intersection of those lines defines the solution. An ill-conditioned system represents two lines which are almost parallel. (It is advisable to try to plot the two lines corresponding to the considered system.)*

Exercise 3 (The effect of fill-in)
`MATLAB` provides a nonsymmetric sparse demonstration matrix `west0479` of dimension 479×479, which can be accessed after executing `load west0479`. Let then rename it as `A=west0479;`.

1. Compute the LU factorization, checking the time it takes to perform it, for example `tic,[L,U,P]=lu(A);toc`.

   Look at `spy` plots of A, PA, L, U and make note of the number of nonzero elements in the triangular factors.

2. Download the MATLAB routine `LUfact_full.m` and make sure you understand the algorithm. It does straightforward LU factorization without pivoting.

   Try to the code on the original matrix A. Is that possible? Why?

   Try to execute if on the permuted matrix $P * A$, where P is the permutation matrix MATLAB has determined in 1. You will soon observe that the execution of the user-written routine takes much more time, even though some efficient MATLAB syntax is used (see the source file). What could the explanation of the above be?

3. Modify A by applying a random column permutation to it.

   ```
   % random column permutation
   col = randperm(479);
   AP  = A(:,col);
   spy(AP)
   ```

   Then repeat 1.

4. Compare the results from 1 and 3.

5. Generate a random right-hand-side vector `b=randn(479,1);`. Solve the system as `x=A\b;`. Compare the solution time with that spent to factorize the matrix.

Exercise 4 (The effect of suitable reordering of the matrix on the number of fillin)
Load the matrices saved in the files `pores_2.mat` and `C29.mat`. Check what kind of matrices are those (`spy, nnz, mesh`(?)).
The task is now to observe the effect of a reordering of the matrix entries prior to the LU-factorization and compare some reordering strategies implemented in MATLAB.
The MATLAB commands which provide reordering vectors are
`symamd, colamd, symmmd, colmmd, symrcm, dmperm`.

1. Check the corresponding MATLAB help for the above orderings

2. Factorize the matrices first without any initial reordering and then after applying three of the reorderings, which are suitable for the matrix you have in hands.

   For each factorization check:

   (a) the time for the factorization,

   (b) the corresponding number of nonzero elements in the L- and U-factors, relative to the total number of nonzero elements in the original matrix;

   Which ordering is best?

Exercise 5 Create a matrix using the routine `Lauchli.m`. The routine is used as follows: `A=Lauchli(N,MU);` i.e., two parameters are required. The created matrix is the $(N+1) \times N$ matrix `[ONES(1,N); MU*EYE(N))]`. If only N is given, then the default value of MU is `sqrt(eps)`.
Since A is rectangular, we can not directly solve systems with it. However, for a given vector b of size$N = 1$ we can find the corresponding Least Square solution
find x which minimizes the quantity $\|b - Ax\|^2$.
One way to obtain the Least Square solution x is to solve the so-called *normal equation* $(A^T A)x = A^T b$.

1. Create at least two matrices (A_1 and A_2) of size, say $N = 100$ and with two different values of MU, one of which to be the default.

2. Take some arbitrary right-hand-side vector $\mathbf{b}$ and solve the system $A_i\mathbf{x} = \mathbf{b},\ i = 1, 2$ using the normal equation. Can you obtain a solution of a "decent" quality? How can one check the quality of the solution obtained when solving the normal equation?

3. Check the condition number of the matrix in the normal equation.

4. Solve the problem in any other way you can think of, using MATLAB's routines. Do you get a better solution? In what way 'better'?

Exercise 6 (A small Google matrix) As a result of the so-called *web-crawling* one creates a matrix representation of the page-references. Based on this information, Google determines the so-called *pageranking* by computing the second largest eigenvalue of those matrices. When you start a Google search, it tells you the size of the actual Google matrix which is used in the search.
For instance, on January 10, 2011 at 9:30, the extra information regarding the search on "Google matrix" was *About 7,600,000 results (0,33 seonds)*
On January 22, 2012, at 22:06, the result was: *About 82,700,000 results (0.29 seconds)*

You can now play with a tiny Google matrix representing the links at Stanford University in 2002. Load a Google matrix by issuing the MATLAB command
`G=loadStanfordMatrix;`
The matrix G is a row-stochastic matrix, which means that all its entries are non-negative and the row-sums are equal to one.
Tasks:

1. Study the matrix - conditioning, structure.

2. Use the routine `GS.m` to orthogonalize some of the columns of G, for example, 5, 10, 20 and perhaps 30 if time permits. For instance, `Q=GS(G,5);`

   Are the computed vectors orthogonal or some orthogonality is lost? The latter can be checked by monitoring the matrix product `Q'*Q`. If the vectors are truly orthogonal, then the product is the identity matrix.

Exercise 7

Load the matrices `PF_54.mat` and `PF_186.mat`. These arise from two consecutive refinements of a model of a real-life problem from multiphase flow, modelled by the so-called Cahn-Hilliard equation.

Study the matrices by all means you know - spectrum, condition number, scaling. Invent an 'exact solution' and corresponding right-hand-side, and try to solve the corresponding systems. Is the obtained solution accurate?

Exercise 8

Consider the Hilbert matrix H of size n defined by

$$H_{i,j} = \frac{1}{i+j+1},\ 1 \leq i,j \leq n$$

This is an example of a very unpleasant matrix for solving equations. To illustrate this we will show that already at the 16*16 size it is almost impossible to get correct solutions for Gaussian Elimination with partial pivoting.

Define a vector `x=ones(n,1)` of length n. (It will be used as the exact solution for the test.) Now define a right-hand-side vector `b = H*x`. We are going to solve the system `A*x=b` using MATLAB's 'backslash' operator and we will compare the obtained solution with the exact solution `x`.

1. Use the MATLAB routine from `Hilbert_matrix.m` which creates a Hilbert matrix of a given dimension. Create a matrix and check its portrait using `mesh`. Check its condition number for a few different sizes of such matrices.

2. Perform the following experiments

```
for n = 2:2:16
    H = Hilbert_matrix(n);
    x = ones(n,1);
    b = H*x;
    y = H\b;
    disp([' Norm of the residual: ' num2str(norm(b-H*y))])
    disp(['Norm of the error:     ' num2str(norm(x-y))])
    figure(1), clf, plot(b-H*y)
    figure(2), clf, plot(abs(x-y))
end
```

   Note how perfect the residual looks and how bad the error becomes.

What is the reason for this behaviour and what can we do to improve the situation?

Used literature:
Carl D. Meyer, *Matrix analysis and applied linear algebra*, SIAM, 2000.
L. Trefethen, D. Bau, *Numerical Linear Algebra*, SIAM, 1997.
J. Leader, *Numerical Analysis and Scientific Computation*, Pearson Education, 2004.
D.S. Watkins, *Fundamentals of Matrix Computations*, J. Wiley & Sins Ltd, 2002.