Numerical methods for large linear algebraic systems

Practical exercises

Dr.ir. C. Vuik

2005

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Introduction

In this note we treat some exercises that are meant as illustration for the theory treated in the course Numerical methods for large linear algebraic systems wi4010. The corresponding matlab routines can be find at:

`http://ta.twi.tudelft.nl/nw/users/vuik/a228/pracfiles.zip`

`or`

`http://ta.twi.tudelft.nl/nw/users/vuik/a228/pracfiles.tar`

```
save the file as: pracfiles.tar
tar -xvf pracfiles.tar
```

Exercise 1
Experiments with the Gaussian elimination method

In this exercise we consider various properties of the Gaussian elimination method. The function `[a,f] = poisson(n1,n2,u1,u2,disc)` is used to construct a matrix, which originates from a discretization of the following equation:

$$-(\frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial y^2}) + u_1 \frac{\partial c}{\partial x} + u_2 \frac{\partial c}{\partial y} = g \text{ where } x, y \in (0,1) \times (0,1),$$

$$c(x,y) = 0 \text{ for } x \in \{0,1\} \text{ or } y \in \{0,1\}.$$

The number of point in the x- and y-direction is denoted by n1, n2 respectively. The value `disc = 'central'` gives a central discretization, whereas `disc = 'upwind'` leads to an upwind discretization. The resulting matrix is a and the right-hand-side vector is f. To get more information of the Matlab programs one can use the `help`-function. Try for instance `help poisson`.

1 a First the fill-in is investigated for small matrices. Therefore the command `[l,u,p] = gauss(a);` can be used. This function computes the LU-decomposition of a with pivoting such that $pa = lu$. Furthermore the number of nonzero elements and the bandwidth in l and u are given. Finally the nonzero structures of the matrices are shown.

Execute the following commands in Matlab:

```
[a,f] = poisson(3,10,0,0,'central');

[l,u,p] = gauss(a);
```

In the subroutine `gauss` the Matlab subroutine `lu` is called. Repeat this exercise with `[a,f] = poisson(10,3,0,0,'central');` and compare both examples.

Related theory: part II, Section 1.2, and 1.7
Exercise aims:

- The matrices L and U are less sparse than A.
- Numbering of the unknowns is important. Numbering such that n1 $\leq$ n2 is better than n1$>$ n2.

1 b In this exercise we take $u1 \neq 0$, or $u2 \neq 0$ and use central differences. It appears that for some choices partial pivoting is used.

Take the following choices:

```
[a,f] = poisson(5,5,0,0,'central');
[l,u,p] = gauss(a);

[a,f] = poisson(5,5,100,0,'central');
```

```
[l,u,p] = gauss(a);

[a,f] = poisson(5,5,0,100,'central');
[l,u,p] = gauss(a);
```

Try also some other values. Warning: for large matrices the CPU-time can increase considerably (`ctrl c` kills a job).

Related theory: part II, Section 1.2, 1.5, and 1.7
Exercise aims:

- Pivoting destroys the non-zero structure of the matrix.
- The amount of fill-in is comparable to that without pivoting.

1 c Do the same exercises as before for upwind differences. Compare the results.

Related theory: part II, Section 1.2, 1.5, and 1.7
Exercise aims:

- The choice of discretization influences the Gaussian elimination algorithm.
- No pivoting is necessary for diagonal dominant matrices.

1 d In this exercise matrices are considered where the amount of fill-in can be very large. The matrices are not related to the Poisson equation. The matrices used here are known as arrowhead matrices. This type of matrix occurs in Domain Decomposition methods. Call `uparrow` and substitute 25 for the dimension. Execute thereafter `[l,u,p] = gauss(a);`. Repeat the exercise for the `downarrow` matrix and compare the results.

Related theory: part II, Section 1.2, and 1.8
Exercise aims:

- Pivoting (or renumbering) can be used to minimize fill-in.

1 e In the following part we investigate the solution of the discretized Poisson equation. Execute `[a,f] = poisson(10,10,0,0,'central');`. The LU-decomposition of the matrix a can be obtained by:

```
[l,u,p] = lu(a);
```

Compute with this LU-decomposition the solution of the linear system $ax = f$. The solution of $ly = f$ can be done by:

```
y = l\f;
```

To check if you have computed the correct solution compute:

```
norm(a*x-f)
```

The resulting solution x can be visualized using the call:

```
plotsol(10,10,x)
```

Related theory: part II, Section 1.2, and 1.3
Exercise aims:

- Compute the solution of a linear system when the LU-decomposition is available.
- Visualization of the solution of the Poisson equation.

1 f Repeat the exercise for

```
[a,f] = poisson(10,10,200,0,'central');
```

Compute again `norm(a*x-f)`. Its value should be less than 10^{-10}. Finally the same problem can be solved using upwind differences. Compare both solutions.

Related theory: part II, Section 1.2, and 1.5
Exercise aims:

- Compute the solution of a linear system when the LU-decomposition with pivoting is available.
- A comparison of the approximations using central and upwind differences.

1 g Finally some experiments are done with iterative improvement. First the matrix and right-hand side are formed by `[a,f] = poisson(5,5,0,0,'central');` Compute the LU-decomposition with

```
[l,u,p] = lu(a);
```

It is not easy to work with single precision numbers in Matlab. To simulate iterative improvement we disturb the matrix l as follows:

```
l = random(l,10^(-5));
```

This function changes the lower triangular matrix l with random numbers of order less than 10^{-5}. Compute `norm(a-lu)` to check that $a \neq lu$ (the matrix u can also be changed). Write a function `iterna(a,l,u,p,f)` which uses the disturbed matrix l (and possible the disturbed matrix u). First construct a termination criterion and make the function such that it stops after 100 iterations.

Related theory: part II, Section 1.2, and 1.5
Exercise aims:

- Get experience with the iterative improvement algorithm.

Exercise 2
Experiments with iterative methods

In this exercise we consider various iterative methods to solve linear systems. We start with systems where the coefficient matrix is symmetric and positive definite. These systems are solved by basic iterative methods and the Conjugate Gradient method. Finally we investigate if these methods can also be applied to non-symmetric systems.

2 a Construct the matrix a and the right-hand-side vector f using the call `[a,f] = poisson(10,11,0,0,'central');` Compute the solution of the linear system with the Gauss-Jacobi method:

```
x = jacobi(a,f,10^-10);
```

Write down the number of iterations for comparison with other methods.

Related theory: part II, Section 2.2
Exercise aims:

- Gauss-Jacobi is a slowly converging method.
- A linear converging iteration method.

2 b First try to modify the program `jacobi.m` to obtain the Gauss-Seidel method. To check your implementation you also use:

```
x = seidel(a,f,10^-10);
```

Compare the results with those obtained in exercise 2 a.

Related theory: part II, Section 2.2
Exercise aims:

- Gauss-Seidel converges two times as fast as Gauss-Jacobi.

2 c In this exercise we use the SOR method. The estimate of the optimal ω is not straightforward. Therefore we do some experiments with various choices of ω. Type

```
x = sor(a,f,10^-10,omega);
```

where `omega` is a real number between 0 and 2. Try to approximate the optimal value. Using the call

```
x = sor(a,f,10^-10,0);
```

an approximation of the optimal ω is calculated from theory. Compare the results.

Related theory: part II, Section 2.2
Exercise aims:

- SOR converges much faster than the previous methods.

- The convergence behavior can be non-linear.
- There are values of ω which leads to somewhat less iterations than the optimal ω calculated from theory.

2 d In this part we consider the Conjugate Gradient method.

```
x = cg(a,f,10^-10);
```

Compare the results with the results obtained with the basic iterative methods.

Related theory: part II, Section 3.2 and 3.3
Exercise aims:

- CG converges very fast.
- It is not necessary to estimate an optimal parameter.
- The convergence behavior is super linear.

2 e In the theory used to analyze CG three quantities are used: $\|x - x_i\|_2$, $\|x - x_i\|_A$, and $\|b - Ax_i\|_2$. These quantities are plotted by the call

```
x = cganal(a,f,10^-10);
```

Compute the condition of a by the Matlab command `cond(a)` and estimate the number of required CG iterations. Compare this with your experiments.

Related theory: part II, Section 3.2 and 3.3
Exercise aims:

- All quantities decrease.
- The number of required estimations obtained from the theory is an upper bound.

2 f There are matrices where the theory for the CG method no longer holds, due to rounding errors. One of them is a discretization of the *bending beam equation*:

$$\frac{d^4c}{dx^4} = f.$$

The matrix can be constructed by `beam;` Take for the dimension 40 (thereafter also try 80 and 160). Solve the system by

```
x = cganal(a,f,10^-10);
```

and observe the results.

Related theory: part II, Section 3.3
Exercise aims:

- $\|x - x_i\|_A$ forms a monotone decreasing sequence.
- $\|x - x_i\|_2$ increases at some iterations.

– CG has not been converged when the number of iterations is equal to the dimension of a.

2 g In the final exercises we investigate if the methods given in Chapter 2 and 3 can be used when the matrices are non-symmetric. Construct the matrix as follows: `[a,f] = poisson(5,5,0.1,0,'central');` and apply Gauss-Jacobi, Gauss-Seidel, SOR, and CG.

Exercise aims:

– All methods converge although the theory for CG is not valid.

2 h Construct the matrix as follows: `[a,f] = poisson(5,5,1,0,'central');` and apply Gauss-Jacobi, Gauss-Seidel, SOR, and CG.

Exercise aims:

– The basic iterative methods converge but CG does no longer converge.

2 i Construct the matrix as follows: `[a,f] = poisson(5,5,100,0,'central');` and apply Gauss-Jacobi, Gauss-Seidel, SOR, and CG.

Exercise aims:

– The iterative methods are divergent.

2 j Construct the matrix as follows: `[a,f] = poisson(5,5,100,0,'upwind');` and apply Gauss-Jacobi, Gauss-Seidel, SOR, and CG.

Exercise aims:

– The basic iterative methods converge.

– The choice of the discretization influences the convergence behavior of an iterative method.