Linear Systems

Exercise 3. Solve the linear system $A\mathbf{x} = \mathbf{b}$ with:

1. the backslash command
2. LU factorization and forward-backward substitutions,

where

$$A = \begin{bmatrix} 1 & 1 & 3 & 0 \\ 2 & 1 & 2 & -1 \\ 3 & 6 & 4 & 1 \\ 1 & -2 & 1 & 3 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 5 \\ 4 \\ 14 \\ 3 \end{bmatrix}$$

Solution Inizialize the matrix and the r.h.s.

```
A=[....];  b=[...];
```

Solve it by backslash: `x=A\b;`
Solve it by LU factorization: `[L,U]=lu(A);`

```
L =
    0.3333    0.2500    1.0000         0
    0.6667    0.7500   -0.2381    1.0000
    1.0000         0         0         0
    0.3333    1.0000         0         0


U =
    3.0000    6.0000    4.0000    1.0000
         0   -4.0000   -0.3333    2.6667
         0         0    1.7500   -1.0000
         0         0         0   -3.9048
```

Why is not `L` lower triangular?

In fact, matlab performs LU with pivoting and `L` holds the effect of pivoting.

If we want to separate the pivoting effects from matrix `L`, the correct comand is: `[L,U,P]=lu(A);`

```
L =
    1.0000         0         0         0
    0.3333    1.0000         0         0
    0.3333    0.2500    1.0000         0
    0.6667    0.7500   -0.2381    1.0000


U =
    3.0000    6.0000    4.0000    1.0000
         0   -4.0000   -0.3333    2.6667
         0         0    1.7500   -1.0000
         0         0         0   -3.9048


P =
     0     0     1     0
     0     0     0     1
     1     0     0     0
     0     1     0     0
```

`P` is a permutation matrix such that $L \cdot U = P \cdot A$ and it takes into account the pivoting of the rows.

In order to solve correctly the system we have:

$$A\mathbf{x} = \mathbf{b} \Leftrightarrow PA\mathbf{x} = P\mathbf{b} \Leftrightarrow LU\mathbf{x} = P\mathbf{b} \Leftrightarrow L\underbrace{(U\mathbf{x})}_{\mathbf{y}} = P\mathbf{b}$$

and then

$$\begin{cases} L\mathbf{y} = P\mathbf{b} \\ U\mathbf{x} = \mathbf{y} \end{cases}$$

The matlab instructions are:

```
[L,U,P]=lu(A);
y=L\(P*b);
x=U\y;
```

Advantages of pivoting

Exercise 4

Let us consider the linear system $A\mathbf{x} = \mathbf{b}$ with

$$A = \begin{pmatrix} 1 & 1 + 0.5 \cdot 10^{-15} & 3 \\ 2 & 2 & 20 \\ 3 & 6 & 4 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} 5 + 0.5 \cdot 10^{-15} \\ 24 \\ 13 \end{pmatrix}$$

1. Solve the linear system by LU factorization without pivoting (use the function `lufact.m`, download it by `dm.ing.unibs.it/gervasio/Nummeth/matlab`) and print the computed solution
2. Solve the linear system by `lu` factorization of matlab (it implements pivoting) and print the computed solution
3. Compare the numerical solution with the exact one $\mathbf{x} = [1, 1, 1]^T$,

Solution

1. By calling `lufact.m` (the syntax is the same as `lu`: `[L,U]=lufact(A)`), we obtain the solution

```
x =
     -4.000000000000003e+00
      6.000000000000000e+00
      1.000000000000000e+00
```

which is very far from the exact solution.
We can compute the relative error between numerical and exact solutions:

```
xex=ones(3,1);
err=norm(x-xex)/norm(xex)
err =
      4.082482904638631e+00
```

it is about 408%.

By solving the linear system with pivoting

```
[L,U,P]=lu(A); y=L\(P*b); x=U\y;
```

we obtain

```
x =
     1.000000000000002e+00
     9.999999999999991e-01
     1.000000000000000e+00
```

Now the relative error w.r.t the exact solution is

```
err=norm(x-xex)/norm(xex)
err =
     1.146633409319802e-15
```

about $10^{-17}\%$.

Remarks.

LU factorization terminates even without pivoting. Neverthless rounding errors propagate very much.

Without pivoting the multipliers are:

$m_{21} = 2$, $m_{31} = 3$ e $m_{32} = -3.4e + 15 \gg 1$. m_{32} is the most responsible of the rounding errors propagation.

```
L =
   1.0000e+00            0            0
   2.0000e+00   1.0000e+00            0
   3.0000e+00  -3.3777e+15   1.0000e+00
```

With pivoting the multipliers are: $m_{21} = 2/3$, $m_{31} = 1/3$ e $m_{32} = 1/2$, all are less than 1.

```
L =
   1.0000e+00            0            0
   6.6667e-01   1.0000e+00            0
   3.3333e-01   5.0000e-01   1.0000e+00
```

It is better to use pivoting in any situation.

Exercise 5. Solve the linear system $A\mathbf{x} = \mathbf{b}$, where A is symmetric and positive definite

$$A = \begin{bmatrix} 5 & 1 & 3 & 0 \\ 1 & 4.2 & 2 & -1 \\ 3 & 2 & 7 & 1 \\ 0 & -1 & 1 & 3 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 5 \\ 4 \\ 14 \\ 3 \end{bmatrix}$$

with a suitable method.

Solution. We can use Cholesky factorization: $R^T R = A$.

$$A\mathbf{x} = \mathbf{b} \Leftrightarrow R^T R\mathbf{x} = \mathbf{b} \Leftrightarrow \begin{cases} R^T\mathbf{y} = \mathbf{b} \\ R\mathbf{x} = \mathbf{y} \end{cases}$$

```
A=[...]; b=[...];
R=chol(A);
y=R'\b; x=R\y
```

We can use also both Gradient and Conjugate Gradient method. Download `grad.m` and `conjgra.m` by the usual web-page.

```
help grad

help conjgra

x0=rand(4,1); kmax=100; tol=1.e-8;

[xg,kkg,errg]=grad(A,b,x0,kmax,tol);

[xcg,kkcg,errcg]=conjgra(A,b,x0,kmax,tol);
```

As predicted by the theory, both methods converge. The conjugate gradient converges exactly in n=4 iterations.

Matlab functions for Conjugate Gradient and Bi-CGStab

The command pcg implements the Conjugate Gradient method

```
[x,flag,relres,iter,resvec]=pcg(A,b,tol,nmax,[],[],x0);
```

Remark. `p` stands for preconditioned and the empty variables `[],[]` are used to pass the preconditioner.

What is a preconditioner?

When $K_2(A) \gg 1$, instead of solving $A\mathbf{x} = \mathbf{b}$ we can solve the equivalent preconditioned system

$$P^{-1}A\mathbf{x} = P^{-1}\mathbf{b}$$

where P is the preconditioner and it is an invertible matrix with the property that $K_2(P^{-1}A) \ll K_2(A)$.

If $K_2(P^{-1}A) \ll K_2(A)$, the preconditioned system is less sensible to rounding errors and an iterative method converges more quickly. The problem is how to build the preconditioner, but this is outside the scope of this lesson.

Like-Conjugate Gradient methods for non-SPD matrices

1. Conjugate Gradient Squared (CGS)

```
[x,flag,relres,iter,resvec]=cgs(A,b,tol,nmax,[],[],x0);
```

2. Bi-Conjugate Gradient (BiCG)

```
[x,flag,relres,iter,resvec]=bicg(A,b,tol,nmax,[],[],x0);
```

3. Bi-Conjugate Gradient Stabilized (BiCGStab)

```
[x,flag,relres,iter,resvec]=bicgstab(A,b,tol,nmax,[],[],x0
```

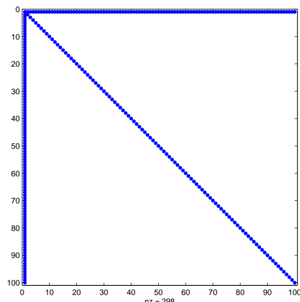
Fast instructions to define matrices and vectors

Let A be the following matrix ($n = 100$):

$$\begin{array}{ll} a_{ii} = i & \text{for } i = 1, ..., n \\ a_{1j} = 1 & \text{for } j = 1, ..., n \\ a_{i1} = 1 & \text{for } i = 1, ..., n. \end{array}$$

How to define and display it

```
n=100;
A=diag(1:n); A(1,:)=ones(1,n); A(:,1)=ones(n,1);
spy(A) % to display the pattern of A
```



The matrix A has only 298 non-null entries (the total number of entries is 10000) and it occupies 80000 Byte, every zero entry occupies 8 Byte

We do not want to store zeros

`double` `sparse` format of Matlab

To store only non-zero entries we can use the `sparse array` of Matlab

```
n=100;
A1=spdiags((1:n)',0,n,n); A1(1,:)=ones(1,n);
A1(:,1)=ones(n,1);
A1 =
   (1,1)        1
   (2,1)        1
   (3,1)        1
   (4,1)        1
   (5,1)        1
....
   (99,99)      99
   (1,100)      1
 (100,100)    100
```

The matrix $A1$ occupies 3980 Byte

sparse format

In displaying the content of a `double sparse` array we have: row-index `i`, column-index `j`, entry `A(i,j)`.

It is possible to convert sparse arrays in full ones and viceversa:
`sparse` converts `double array` in `double (sparse) array`
`full` converts `double (sparse) array` in `double array`
If `A1` holds a sparse array and `A` a full array:

```
AF1=full(A1);  % AF1 is full
AS=sparse(A);  % AS is sparse
```

spdiags command

Exercise 6. Solve the linear system $A\mathbf{x} = \mathbf{b}$ of size n with

$$A = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & -1 & 2 & -1 \\ 0 & \dots & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 0.1 \\ h^2 \\ \vdots \\ h^2 \\ 0.5 \end{bmatrix},$$

where $h = 1/(n-1)$.

Solution. The matrix A is not symmetric, due to the first and last rows. At the same time, the first and last rows say that $x_1 = 0.1$ and $x_n = 0.5$. We can modify the system:

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & -1 & 2 & -1 \\ 0 & \dots & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} 0.1 \\ h^2 \\ \vdots \\ h^2 \\ 0.5 \end{bmatrix}$$

in

$$\begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 2 & -1 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & -1 & 2 & 0 \\ 0 & \dots & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} 0.1 \\ h^2 \\ \vdots \\ h^2 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 1 \\ -1 \\ \vdots \\ 0 \\ 0 \end{bmatrix} x_1 - \begin{bmatrix} 0 \\ 0 \\ \vdots \\ -1 \\ 1 \end{bmatrix} x_n$$

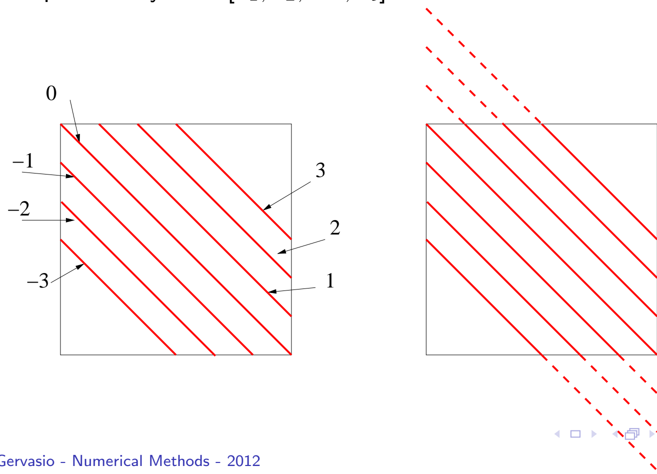
We replace $x_1 = 0.1$ and $x_n = 0.5$ on the right and we eliminate both first and last rows, first and last column:

$$\begin{bmatrix} 2 & -1 & \dots & & 0 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ 0 & & \dots & -1 & 2 \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \\ \vdots \\ x_{n-2} \\ x_{n-1} \end{bmatrix} = h^2 \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0 \\ \vdots \\ 0 \\ 0.5 \end{bmatrix}$$

and now A is symmetric. It is also positive definite. We solve the system by `pcg.m`

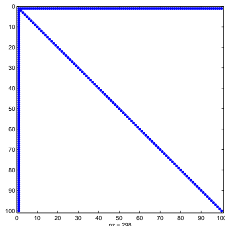
```
n=30; e=ones(n,1);
A=spdiags([-e,2*e,-e],(-1:1),n-2,n-2);
spy(A)
```

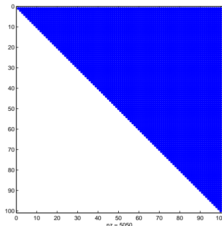
The command `A = spdiags(B,D,n,m)` creates a sparse matrix A of size $n \times m$. The column vectors of the array B are put inside the diagonals of A specified by $D = [d_1, d_2, \ldots, d_t]$



```
h=1/(n-1);
b=h^2*ones(n-2,1); b(1)=b(1)+0.1; b(n-2)=b(n-2)+0.5;
x0=rand(n-2,1); kmax=100; tol=1.e-12;
[x,flag,relres,iter,resvec]=pcg(A,b,tol,kmax,[],[],x0);
```

The fill-in phenomenon





pattern of A pattern of U

The fill-in occurs in direct methods during the elimination (or factorization). Even if the matrix A has a few non-null entries, the matrix $U =$ of LU factorization (but also $\tilde{A}$ of GEM) can have very high density. All the direct methods act on the matrix and they transofrm it in a denser matrix.

On the contrary iterative methods do not modify the matrix, then no fill-in occurs in iterative methods.