Prof. W. Gander **Masaryk University Fall 2014**

Linear Equations

1. Write a function `x=forwards(L,b)` to solve the system $L\mathbf{x} = \mathbf{b}$ with the lower diagonal matrix L by forward substitution using the SAXPY variant.

   Solution:

```
function x=forwards(L,b)
% FORWARDS forwardsubstitution in Lx=b
% solution of exercise
n=length(b);
for i=1:n;
   x(i)=b(i);
   b(i+1:n)=b(i+1:n) -x(i)*L(i+1:n,i);
end
x=x(:);
```

2. Consider the linear system $A\mathbf{x} = \mathbf{b}$ with

$$A = \begin{pmatrix} 1 & 2 & -2 & -6 \\ -3 & -1 & -2 & \alpha \\ -4 & 3 & 9 & 16 \\ 5 & 7 & -6 & -15 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

   The element $a_{24} = \alpha$ has been lost. Assume, however, that before when α was available, the solution with MATLAB turned out to be

```
> x=A\b
x =
   1.0e+15 *

    0.7993
   -0.3997
    1.1990
   -0.3997
```

   Can you determine with this information the missing integer matrix element $\alpha = a_{24}$?

   Solution: The computed solution is large, it solves numerically

$$A\mathbf{x} = \mathbf{b}.$$

If we divide both sides by $1.1990 * 1.0e{+}15$ we get with

$$A \begin{pmatrix} 0.6666 \\ -0.3334 \\ 1.0000 \\ -0.3334 \end{pmatrix} = \frac{1}{1.1990}\mathbf{b} \times 10^{-15} \approx 0$$

Thus

$$\mathbf{x}_h = \begin{pmatrix} 0.6666 \\ -0.3334 \\ 1.0000 \\ -0.3334 \end{pmatrix} = \begin{pmatrix} 2/3 \\ -1/3 \\ 1 \\ -1/3 \end{pmatrix}$$

is a solution of the homogeneous system $A\mathbf{x} = 0$! Inserting $\mathbf{x}_h$ in the second equation we obtain

$$-3 \cdot 2/3 + 1/3 - 2 - \alpha/3 = 0$$

and thus $\alpha = -11$.

3. Use the function `EliminationGivens` to investigate if the two lines g and h intersect. If they don't intersect, compute their distance.

$$g : \mathbf{x} = \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix} + r \begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix}, \qquad h : \mathbf{x} = \begin{pmatrix} 3 \\ -7 \\ 2 \end{pmatrix} + s \begin{pmatrix} 3 \\ -1 \\ -3 \end{pmatrix},$$

Solution:

```
% lines in 3d

% g:
P=[1 2 2]';
d1=[1 3 1]';
% h:
Q=[3 -7 2]';
d2=[3 -1 -3]';

% Equate g and h to find intersection
A=[d1 -d2]; b=Q-P;

% Solve for parameters with EliminationGivens
% p=[r,s]'
[p,distance]=EliminationGivens(A,b)

Residual=b-A*p
NormResidual=norm(Residual)

closestPoints=[P+p(1)*d1, Q+p(2)*d2]

dist=norm(P+p(1)*d1-(Q+p(2)*d2))
```

```
p =
   -2.1500
   -0.4500
distance =
    4.9497
Residual =
    2.8000
   -2.1000
    3.5000
NormResidual =
    4.9497
closestPoints =
   -1.1500    1.6500
   -4.4500   -6.5500
   -0.1500    3.3500
dist =
    4.9497
```

Note that after the solution with Givens the norm of the residual is nonzero. This means that the lines do not intersect. The norm of the resulting residual is the distance between the nearest points on the lines. Using the parameter solutions we construct the nearest points and check that the distance is the same.

4. Consider the linear system of equations $A\mathbf{x} = \mathbf{b}$ with the matrix

$$A = \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix} \tag{1}$$

and choose $\mathbf{b}$ such that the exact solution is the vector `[1:n]`.

Write an experimental program for $n = 1000$ using first the Jacobi method. Perform 100 Jacobi iteration steps and estimate the spectral radius using $\mathbf{u}_{k+1} = M^{-1}N\mathbf{u}_k$ and $\rho \approx \|\mathbf{u}_{k+1}\|/\|\mathbf{u}_k\|$.

Compare the estimated spectral radius to the true one.

Then compute with the estimated spectral radius according to the theory of David Young the optimal over-relaxation parameter ω and continue for 100 more iteration steps with SOR.

Compute in each step the true error $\mathbf{e}_k = \mathbf{x} - \mathbf{x}_k$ and store its norm in a vector. Finally use `semilogy` and plot the error norm as a function of the iteration steps. You should notice a better convergence with SOR.

Solution: The MATLAB program is straightforward:

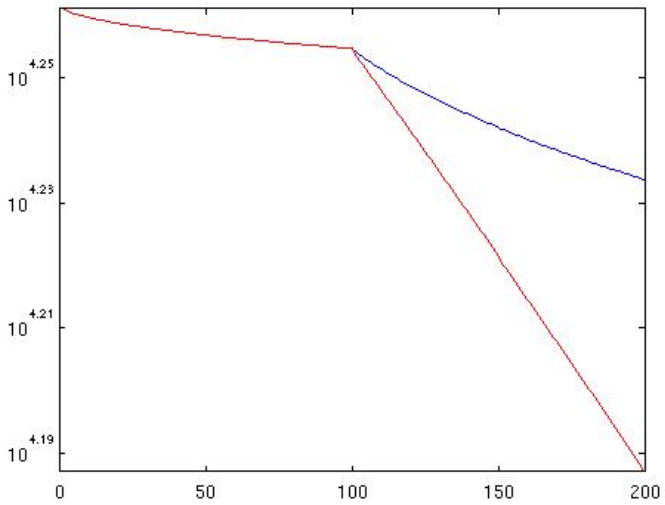
```
clear,clf,clc
format long e
```

```
% define the matrix and the splitting
  n=1000
  h=ones(n-1,1);
  M=2*eye(n); N=diag(h,-1)+diag(h,1); A=M-N;

% define exact solution and right hand side
  xe=[1:n]'; b=A*xe;

% 100 Jacobi Iteration steps
% store error in rr
  rr=[];
  x=zeros(size(xe));  %start vector
  xnew=M\(N*x+b);
  u=xnew-x;           % difference

  x=xnew;
  for k=1:100
    xnew=M\(N*x+b);
    unew= xnew-x;
    rho=norm(unew)/norm(u);
    x=xnew;
    u=unew;
    rr=[rr, norm(x-xe)];
  end

  rhoestimate=rho;
  rhoex=max(abs(eig(M\N)));
  disp('convergence rate Jacobi'), [rhoestimate rhoex]

% optimal over relaxation parameter
  omega=2/(1+sqrt(1-rhoestimate^2));
  omegaex=2/(1+sqrt(1-rhoex^2));
  disp('optimal over relaxation parameter')
  [omega omegaex]

  xx=x; rrx=rr;  % save values after Jacobi

% Now continue with SOR with estimates rho
  D=M;  L=-tril(N);  U=-triu(N);
  for k=1:100
    xnew=(D+omega*L)\((-omega*U+(1-omega)*D)*x+omega*b);
    unew= xnew-x;
    rho=norm(unew)/norm(u);
    x=xnew;
    u=unew;
    rr=[rr, norm(x-xe)];
  end
  semilogy(rr)
  hold
```

```
  drawnow;
  pause

% Now continue with SOR with exact rho
  x=xx; rr=rrx;  omega=omegaex;
  for k=1:100
    xnew=(D+omega*L)\((-omega*U+(1-omega)*D)*x+omega*b);
    unew= xnew-x;
    rho=norm(unew)/norm(u);
    x=xnew;
    u=unew;
    rr=[rr, norm(x-xe)];
  end
  semilogy(rr,'r')
```

We obtain the following results:

	estimated	exact
ρ	0.992546479945137	0.999995075056662
ω	1.782743042711722	1.993742739997419



We see that SOR clearly improves convergence. The upper curve is the result with the estimated ρ, the lower with the exact value. The over-relaxation factor ω with the exact ρ is much closer to 2 and the convergence is better.

5. Solve the same system using the conjugate gradient method. Modify our function `CG` such that the norm of the error is stored for every step. For this you need to add the exact solution to the input parameter list.

```
[x,Errors]=conjgradient(A,b,x0,xexact,m);
```

(a) Apply 200 steps of CG and plot the norm of the errors and compare it to SOR.

(b) Plot the error and the residuals for $m = 1000$

Solution:

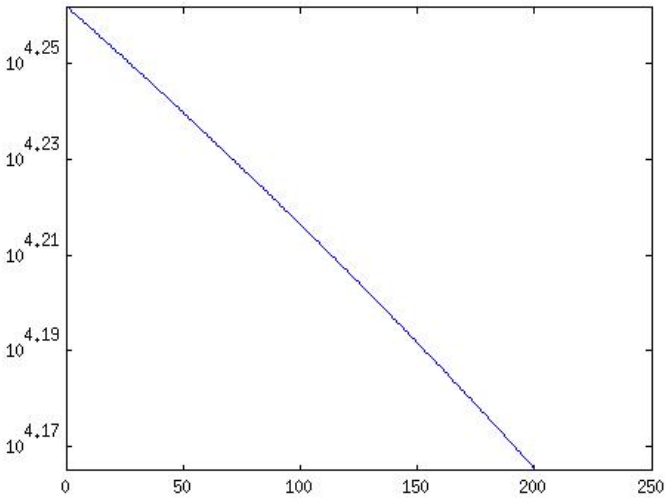
(a) The modified function becomes

```
function [x,Errors]=conjgradient(A,b,x0,xexact,m);
%
x=x0; r=b-A*x; p=r;
Errors=norm(x-xexact);
oldrho =r'*r;
for k=1:m
  Ap = A*p;
  alpha(k) = oldrho/(p'*Ap);
  x = x+alpha(k)*p;
  r = r-alpha(k)*Ap;
  rho= r'*r;
  beta(k) = rho/oldrho;
  oldrho=rho;
  p = r+beta(k)*p;
Errors=[Errors, norm(x-xexact)];
end
```

and the main program is

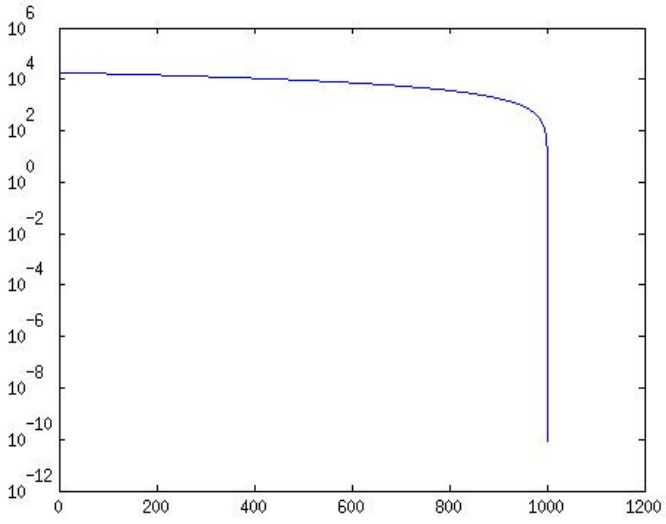
```
clear,clf,clc
format long e
% define the matrix and the splitting
  n=1000
  h=ones(n-1,1);
  M=2*eye(n); N=diag(h,-1)+diag(h,1); A=M-N;

% define exact solution and right hand side
  xe=[1:n]'; b=A*xe;

x0=0*xe;
m=200;
[x,Errors]=conjgradient(A,b,x0,xe,m);
semilogy(Errors)
```

The resultin plot of the error is



The error decreases more rapidly than with SOR.

(b) If we perform 1000 steps we get the plot



We see that for this matrix CG behaves essentially like an explicit method – it needs $n = 1000$ steps to converge. The matrix of the linear system is tridiagonal and all its eigenvalues are simple.

CG converges well as an iterative method with a good preconditioner. The preconditioner should cluster the eigenvalues of the matrix. Then we get fast convergence.

6. A linear system with the matrix (1) could be preconditioned with the matrix

$$M = \begin{pmatrix} 1 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix} = FF^\top \quad \text{with} \quad F = \begin{pmatrix} 1 & & & \\ -1 & 1 & \ddots & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{pmatrix}$$

Suppose you are solving the preconditioned system $F^{-1}AF^{-\top}(F^{\top}\mathbf{x}) = F^{-1}\mathbf{b}$ with CG. How many iteration steps are at most necessary? Justify your claim by applying Corollary 2.1.

Solution: The corollary states that if A has only $m \leq n$ different eigenvalues, then CG converges in at most m steps.

Since $FF^{\top} = A - N$ where $N = \mathbf{e}_1\mathbf{e}_1^{\top}$ is a rank one matrix, we get by replace the matrix A by

$$F^{-1}AF^{-\top} = I + F^{-1}NF^{-\top}$$

a rank one modification if the identity matrix. This matrix has only two different eigenvalues. Thus CG will solve the preconditioned system

$$F^{-1}AF^{-\top}\mathbf{y} = F^{-1}\mathbf{b}$$

in two steps.

We verify this with a numerical example.

```
format compact
n=10
A=2*eye(n)-diag(ones(n-1,1),1)-diag(ones(n-1,1),-1);
xe=[1:n]'; b=A*xe;                % generate system with solution
F=eye(n)-diag(ones(1,n-1),-1);
[Y,R] = CG(F\A/F',F\b,0*xe,2)     % solve preconditioned system
x=F'\Y                            % transform variables

n =
    10
Y =
         0         0   -1.0000
         0         0   -1.0000
         0         0   -1.0000
         0         0   -1.0000
         0         0   -1.0000
         0         0   -1.0000
         0         0   -1.0000
         0         0   -1.0000
         0         0   -1.0000
         0    5.5000   10.0000
R =
         0   -5.5000         0
         0   -5.5000         0
         0   -5.5000         0
         0   -5.5000         0
         0   -5.5000         0
         0   -5.5000         0
         0   -5.5000         0
         0   -5.5000         0
         0   -5.5000         0
```

```
   11.0000         0         0
x =
         0    5.5000    1.0000
         0    5.5000    2.0000
         0    5.5000    3.0000
         0    5.5000    4.0000
         0    5.5000    5.0000
         0    5.5000    6.0000
         0    5.5000    7.0000
         0    5.5000    8.0000
         0    5.5000    9.0000
         0    5.5000   10.0000
```

Indeed we notice that the residual is zero after two steps and that we obtain the exact solution.