University of Science
Faculty of Mathematics and Computer Science

Assignment #1

Professor: *Laurence Halpern - Juliette Ryan*

Student name: *Nguyen Tu Huy*

Course: *PUF - High Performance Computing*
Due date: *November 29th, 2022*

Consider problem: Let A be the matrix of size 100 : $A = SDS^{-1}$ with $S = (1, \beta)$ an upper bidiagonal matrix. $\beta = 0.9$ and $D = \alpha * diag(1, 2, ..., 100)$ and $\alpha = 2$.

To check the validity of your algorithms, set :

- $x_{exact} = rand(100, 1)$
- $b = Ax_{exact}$
- if x is your computed solution, check the error $\|x - x_{exact}\|_2$

1. Exercise 1

Write A as $A = D - E - F$.
Use Matlab functions *diag, triu, tril.*
Solve $Ax = b$ (and check) with the following methods and comment the efficiency of these different methods.

- Jacobi
- Gauss Seidel $(D - E)x = Fx + b$
- Backward Gauss Seidel $(D - F)x = Ex + b$
- Simple projection method : Steepest Descent
- Successive Over Relaxation (SOR) $\omega A = (D - \omega E) - (\omega F + (1 - \omega)D)$.
  Study convergence as a function of ω

Answer

Jacobi method:

```
function [x,res,it,time]=Jacobi(A,b,x0,tol,itmax)
% JACOBI  Resolution of linear systems by Jacobi method.
%   [x,res]=JACOBI(A,b,x0,tol,itmax) solves Ax=b by
%   Jacobi using x0 as initial guess. The algorithm
%   stops when maxiter operations have been performed,
%   or when the L2 norm of residual becomes smaller than tol.
```

```
%   The output x is the approximation of the solution,
%   and res is a vector containing the history of the residuals.

D = diag(diag(A));
E = -tril(A) + D;
F = -triu(A) + D;

x = x0;
res(1) = norm(b-A*x,2);
it=1; time = cputime();
while(it<=itmax && res(it)>tol)
    x = inv(D)*(b + (E+F)*x);
    res(it+1) = norm(b-A*x,2);
    it = it+1;
end
time = cputime()-time;
end
```

Gauss Seidel method:

```
function [x,res,it,time]=GaussSeidel(A,b,x0,tol,itmax)
% GAUSSSEIDEL Resolution of linear systems by Gauss-Seidel
%   [x,res]=GAUSSSEIDEL(A,b,x0,tol,itmax) solves Ax = b by
%   Gauss-Seidel using x0 as initial guess. The algorithm
%   stops when maxiter operations have been performed,
%   or when the L2 norm of residual becomes smaller than tol.
%   The output x is the approximation of the solution,
%   and res is a vector containing the history of the residuals.

D = diag(diag(A));
E = -tril(A) + D;
F = -triu(A) + D;

x = x0;
res(1) = norm(b-A*x,2);
it = 1; time = cputime();
while(it <= itmax && res(it)>tol)
    x = inv(D-E)*(b + F*x);
    res(it+1)=norm(b-A*x,2);
    it=it+1;
end
time = cputime()-time;
end
```

Backward Gauss Seidel method:

```
function [x,res,it,time]=Backward_GaussSeidel(A,b,x0,tol,itmax)
% BACKWARD GAUSSSEIDEL Resolution of linear systems
%   [x,res]=BACKWARD_GAUSSSEIDEL(A,b,x0,tol,itmax) solves Ax = b by
%   Gauss-Seidel using x0 as initial guess. The algorithm
%   stops when maxiter operations have been performed,
%   or when the L2 norm of residual becomes smaller than tol.
%   The output x is the approximation of the solution,
%   and res is a vector containing the history of the residuals.

D = diag(diag(A));
E = -tril(A) + D;
F = -triu(A) + D;

x = x0;
```

```
res(1) = norm(b-A*x,2);
it = 1; time = cputime();
while(it <= itmax && res(it)>tol)
    x = inv(D-F)*(E*x+b);
    res(it+1)=norm(b-A*x,2);
    it=it+1;
end
time = cputime()-time;
end
```

Steepest Descent:

```
function [x,res,it,time]=Steepest_Descent(A,b,x0,tol,itmax)
% STEEPEST DESCENT Resolution of linear systems
%   [x,res]=Steepest_Descent(A,b,x0,tol,itmax) solves Ax = b by
%   Steepest Descent using x0 as initial guess. The algorithm
%   stops when maxiter operations have been performed,
%   or when the L2 norm of residual becomes smaller than tol.
%   The output x is the approximation of the solution,
%   and res is a vector containing the history of the residuals.

x = x0;
r = b - A*x;
res(1) = norm(b-A*x,2);
it = 1; time = cputime();
while(it <= itmax && res(it)>tol)
    alpha = (r'*r)/(r'*A*r);
    x = x + alpha*r;
    r = b - A*x;
    res(it+1)=norm(b-A*x,2);
    it=it+1;
end
time = cputime()-time;
end
```

Successive Over Relaxation (SOR) method:

```
function [x,res,it,time]=SOR(A,b,relax,x0,tol,itmax)
% SOR Resolution of linear systems by overrelaxation method
%   [x,res]=SOR(A,b,relax,x0,tol,itmax) solves Ax=b by
%   SOR with parameter relax  using x0 as initial guess. The algorithm
%   stops when maxiter operations have been performed,
%   or when the L2 norm of residual becomes smaller than tol.
%   The output x is the approximation of the solution,
%   and res is a vector containing the history of the residuals.

E = tril(A,-1);
F = triu(A,1);
D = diag(diag(A));

x = x0;
res(1)=norm(b-A*x,2);
it=1; time = cputime();
while it<=itmax && res(it)>tol
    x = (D/relax+E)\(((1-relax)/relax*D-F)*x+b);
    res(it+1) = norm(b-A*x,2);
    it = it+1;
end
time = cputime()-time;
end
```

We apply different methods above to solve $Ax = b$.

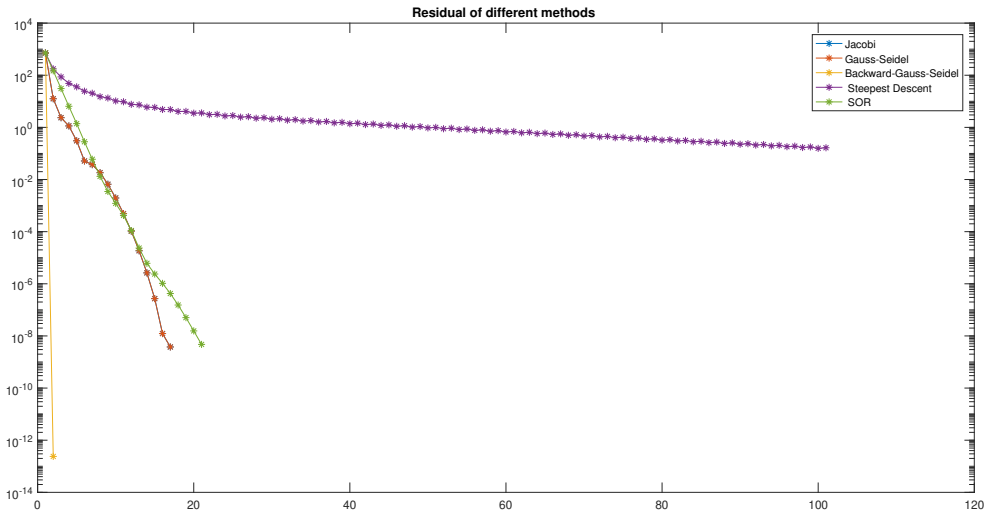

Figure 1: Comparison of residual value of different methods

Table compare number of iteration, error and time compute of difference methods:

	Jacobi	Gauss Seidel	Backward GS	Steepest Descent	SOR (w = 1.2)
iteration	17	17	2	101	21
error	3.6746e-09	3.6746e-09	2.8832e-15	0.049186	3.5914e-09
time	0.0086781	0.010933	0.0077338	0.013098	0.010401

Commend: Among those methods, Backward Gauss Seidel show the best result.

In 1947, Ostrowski proved that if A is **symmetric** and **positive-definite**. The SOR method convergence if $0 < w < 2$.

And the result of table below show that for the case $w = 0$ and $w = 2$, the solution tend to infinity or not define.

Difference type of w in Successive Over Relaxation (SOR) method

SOR method			
relax	error	iteration	time
0.00	NaN	2	0.0050315
0.10	0.02244	101	0.0059529
0.20	1.2506e-05	101	0.0087116
0.30	1.1534e-09	101	0.0055591
0.40	5.3795e-13	93	0.0049079
0.50	4.3825e-13	71	0.0037661
0.55	3.7582e-13	63	0.0036237
0.60	3.8207e-13	56	0.003665
0.65	3.8992e-13	50	0.0030821
0.70	3.3902e-13	45	0.0025345
0.75	4.964e-13	40	0.0033605
0.80	4.6651e-13	36	0.0029168
0.85	1.953e-13	33	0.0019555
0.90	3.6144e-13	29	0.0015899
0.95	1.8288e-13	26	0.0019613
1.00	3.8022e-13	22	0.0014146
1.05	6.9432e-14	19	0.0011066
1.10	2.1207e-13	21	0.0012507
1.15	1.3179e-13	25	0.0026165
1.20	4.5089e-13	28	0.0017841
1.25	2.1345e-13	33	0.0018032
1.30	2.2188e-13	38	0.002525
1.35	4.4248e-13	43	0.0022318
1.40	3.893e-13	50	0.0027628
1.45	4.3487e-13	58	0.0028474
1.50	3.9608e-13	68	0.0035762
1.60	4.4941e-13	95	0.0066112
1.70	8.2683e-08	101	0.005855
1.80	0.029611	101	0.004917
1.90	2468.9	101	0.0053943
2.00	6.2484e+07	101	0.0050444

Table 1: Comparison of difference value of w

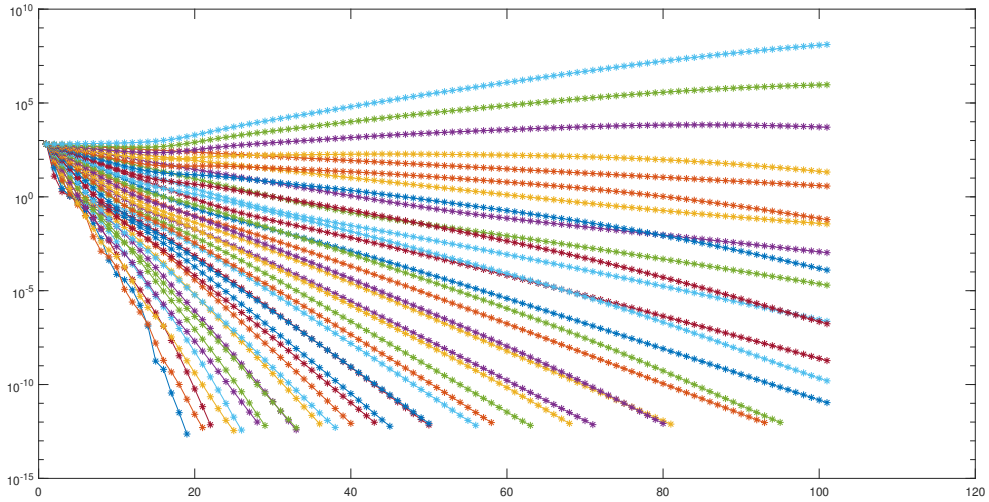

Figure 2: Comparison different value of w in SOR method

Commend:

- For difference value of w among $(0, 2)$, the number iteration of convergence is difference.
- The optimal value of w in $(1, 1.15)$

2. Exercise 2

- Can you solve $Ax = b$ with a *Conjugate gradient*? Why?
- Modify A and b so that you can find the solution x with a *Conjugate gradient* and solve it.
- Solve the modified problem with *Jacobi*, *Gauss Seidel*, *Steepest Descent*, *SOR* and compare the costs

Answer

Since A is not a symmetric positive definite matrix, we can not apply a *Conjugate gradient*. Indeed, *Conjugate gradient* work if we ensure 2 factor:

- r^0(the initial residual) generates the orthonormal basis of *Krylov space*, $\mathcal{K}_m = \text{vec}(r^0, Ar^0, ..., A^{n-1}r^0)$. This is guaranteed to happen if A is symmetric.
- The CG method to work, it is required that the Cholesky factorization (or more precisely, the non-pivoted LDL^T) of the tridiagonal matrix generated by the Lanczos method exists. The non-existence of this factorization is related to the fact that if a symmetric/Hermitian matrix is indefinite, then the coefficients α_k in the CG implementation can be undefined (as $x^T Ax$ may be zero for some nonzero x). The non-pivoted LDL^T factorization, however, is guaranteed to exist if the matrix is positive definite.

To ensure the condition A is symmetrix positive definite, we modify the matrix A:

$$A = A^T A$$

. Since $A^T A$ is symmetrix positive definite, we can apply the *Conjugate gradient* method.

Conjugate Gradient method:

```
function [x,res,it,time] = CG(A,b,x0,tol,itmax)
% CG conjugate gradient method
% [X,R,a,b]=CG(A,b,x0,n) computes an approximation for the solution
% of the linear system Ax=b performing n steps of the conjugate
% gradient method starting with x0, and returns in res the norm of the
   residual
x = x0; r = b-A*x;
res(1) = norm(r,2);
p = zeros(size(b));
rho = 1;
it = 1; time = cputime();
while(it<=itmax && res(it)>tol)
    rhoold = rho; rho = r'*r;
    beta = rho/rhoold;
    p = r + beta*p;
    z = A*p;
    alpha = rho/(p'*z);
    x = x + alpha*p;
    r = r - alpha*z;
    res(it+1) = norm(r,2);
    it = it+1;
end
time = cputime()-time;
```

We apply CG method and those methods above to solve Ax = b.

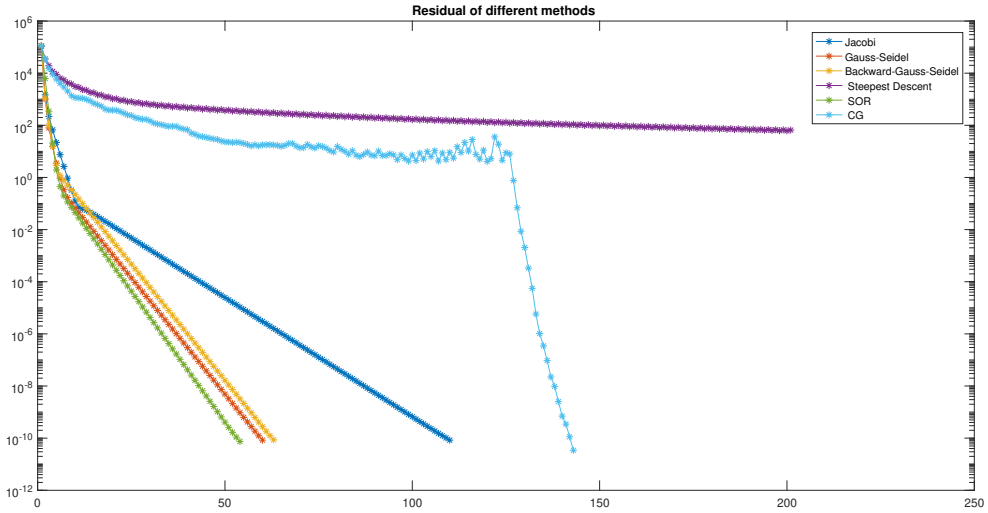

Figure 3: Comparison residual of different method

Table compare number of iteration, error and time compute of difference methods:

	Jacobi	Gauss Seidel	Backward GS	Steepest Descent	SOR ($w = 1.05$)	Conjugate gradient
iteration	110	60	63	201	54	143
error	1.0596e-11	2.7996e-11	4.9453e-12	0.85728	2.545e-11	5.256e-15
time	0.017423	0.069534	0.078796	0.019328	0.01418	0.01243

Commend: Among those method,

- SOR method (w = 1.05) convergence with less iteration.
- Conjugate gradient compute with fastest time.

If we consider tol $= 10^{-14}$**,**

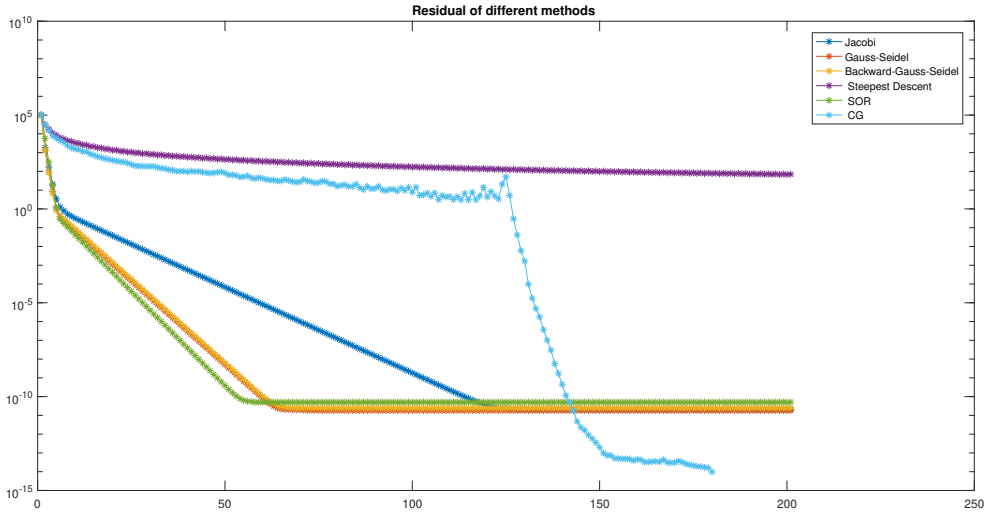

Figure 4: Comparison residual of different method

	Jacobi	Gauss Seidel	Backward GS	Steepest Descent	SOR ($w = 1.05$)	Conjugate gradient
iteration	201	201	201	201	201	180
error	4.0565e-15	4.2935e-15	4.5639e-15	1.1858	3.8849e-15	4.9558e-15

Commend: Only Conjugate Gradient give residual less than 10^{-14} before it reach itmax.

3. Exercise 3

Apply these different solvers to the Laplace equation discretised with 2nd order Finite Differences in dimension 1 and 2 (*See Prof. Halpern's course*) with Dirichlet boundary conditions. Check that the discretisation provides a second order numerical solution. (*Use manufactured solutions*).

The **Laplace equation** discretised with 2nd order Finite Differences in dimension 1

```
function A=lap1d(n)
% lap1d one dimensional finite difference approximation
%   A=lap1d(n) computes a sparse finite difference
%   approximation of the one dimensional operator -Delta on the
%   domain Omega=(0,1) using n interior points

h = 1/(n+1);
e = ones(n,1);
A=spdiags([-e/h^2 2/h^2*e -e/h^2],[-1 0 -1],n,n);
```

Since other method do not convergence, only *Conjugate Gradient* method convergence

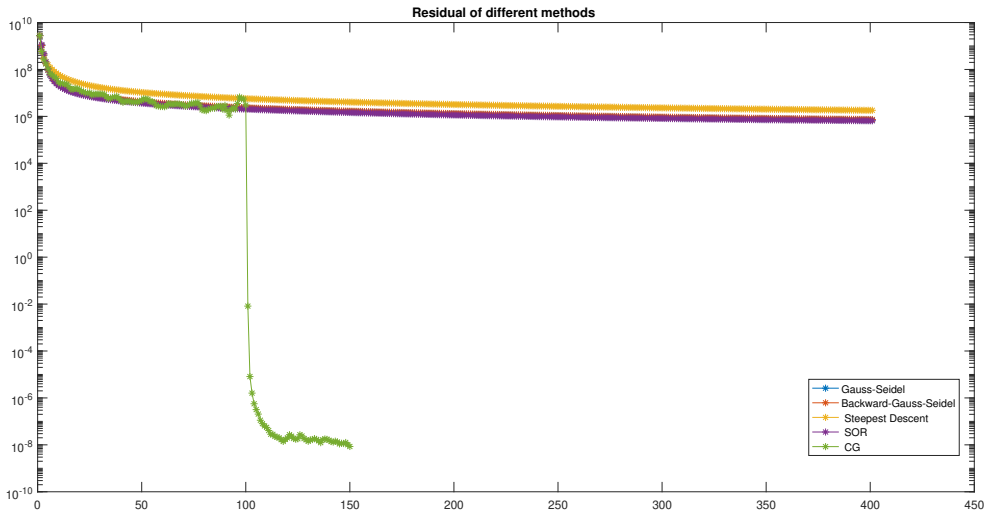

Figure 5: Compare difference method for solving equation $Ax = b$ apply to PDE 1D with Dirichlet boundary problem

	Jacobi	Gauss Seidel	Backward GS	Steepest Descent	SOR ($w = 1.05$)	Conjugate gradient
iteration	401	401	401	401	401	150
error	4.3279	4.0309	4.0612	4.3234	3.9806	1.7268e-13

The **Laplace equation** discretised with 2nd order Finite Differences in dimension 2

```
function A=lap2d(nx,ny)
% A=lap2d(nx,ny) matrix of -delta in 2d on a grid
% of  nx  internal points in x and ny  internal points in y
```

```
% numbered by row. Uses the function kron of matlab
Dxx=lap1d(nx);
Dyy=lap1d(ny);

A=kron(speye(size(Dyy)),Dxx) + kron(Dyy,speye(size(Dxx)));
```

Since other method do not convergence, only *Conjugate Gradient* method convergence (by means error less than 10^{-10} and iteration less than 401)

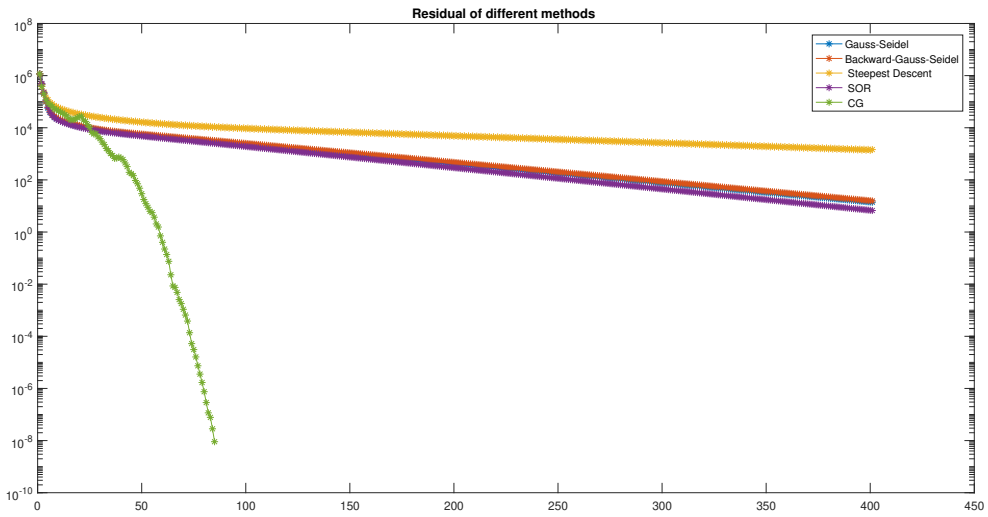

Figure 6: Compare difference method for solving equation $Ax = b$ apply to PDE 2D with Dirichlet boundary problem

	Jacobi	Gauss Seidel	Backward GS	Steepest Descent	SOR ($w = 1.05$)	Conjugate gradient
iteration	401	401	401	401	401	150
error	2.62e+81	4.6352e-3	5.3741e-3	0.35277	2.2081e-3	4.7675e-14