University of Science
Faculty of Mathematics and Computer Science

Worksheet #2

Professor: *Laurence Halpern - Juliette Ryan*

Student name: *Nguyen Tu Huy*

Course: *PUF - High Performance Computing*
Due date: *December 01th, 2022*

Let A be the

a) matrix defined in worksheet 1

b) The 1D Finite Difference Matrix

c) Matrices defined in Matrix Market.

1. Exercise 1

In the restarting **gmres.m** you received , add the missing lines that transform the **Hessenberg** matrix into an upper matrix by applying Givens rotation and which also provides the error at iteration $j+1$.
Check that your restarting **gmres.m** is correct using manufactured solutions.

- First , use a restart parameter that is greater than the matrix dimension. What does this imply?
- Analyse in terms of computing and memory storage , the effect of the restarting parameter.

Comment the efficiency of gmres compared to the previous iterative methods worksheet 1).

GMRES.m:

```
%*************************************************
function [x,error,iter] = GMRES(A,b,x,m,itmax,epsi)
%*************************************************
% gmres.m solves the linear system Ax=b
% using the Generalized Minimal residual ( GMRESm ) method with restarts
% input   A        REAL nonsymmetric positive definite matrix
%         x        REAL initial guess vector
%         b        REAL right hand side vector
%         m        INTEGER number of iterations between restarts
%         itmax    INTEGER maximum number of iterations
%         epsi      REAL error tolerance
%
% output  x        REAL solution vector
%         error    REAL error norm
%         iter     INTEGER number of iterations performed

```

```
% initialization

normb = norm(b);
if  ( normb == 0.0 )
    normb = 1.0;
end
% residual
r = b - A*x;
error(1) = norm(r)/normb;
if ( error(1) < epsi )
    return;
end
[n,n] = size(A);
V(1:n,1:m+1) = zeros(n,m+1);    % Vm+1=[Vm|qm+1]
H(1:m+1,1:m) = zeros(m+1,m);    % Hessenberg matrix
cs(1:m) = zeros(m,1);
sn(1:m) = zeros(m,1);

e1    = zeros(n,1);             % basic vector
e1(1) = 1.0;
iter = 1;                       % step of iterator
itt = 1;
while iter <= itmax             % begin iteration
    r = b-A*x;
    V(:,1) = r/norm(r);
    s = norm(r)*e1;
    for j = 1:m                 % construct orthonormal
        itt = itt+1 ;           % basis using Gram-Schmidt
        w = A*V(:,j);
        for i = 1:j
            H(i,j)= w'*V(:,i);
            w = w - H(i,j)*V(:,i);
        end
        % size(H)
        H(j+1,j) = norm(w);
        V(:,j+1) = w/H(j+1,j);
% We tranform the Hessenberg matrix H into a triangular matrix by
   applying Givens rotation
        for k = 1:j-1           % apply Givens rotation
            temp     =  cs(k)*H(k,j) + sn(k)*H(k+1,j);
            H(k+1,j) = -sn(k)*H(k,j) + cs(k)*H(k+1,j);
            H(k,j)   = temp;
        end
        cs(j) = H(j,j)/sqrt(H(j,j)^2 + H(j+1,j)^2);
        sn(j) = H(j+1,j)/sqrt(H(j,j)^2 + H(j+1,j)^2);
        temp   = cs(j)*s(j);     % approximate residual norm
        s(j+1) = -sn(j)*s(j);
        s(j)   = temp;
        H(j,j) = cs(j)*H(j,j) + sn(j)*H(j+1,j);
        H(j+1,j) = 0.0;
        % Which also provides the error at iteration j+1

        error(itt+1) = abs(s(j+1)) / normb;
        if ( error(itt+1) <= epsi )     % update approximation
            y = H(1:j,1:j)\s(1:j);                      % and exit
            x = x + V(:,1:j)*y;
            % error(i+1) = abs(s(i+1)) / bnrm2;
            break;
```

```
        end
    end

    if ( error(itt+1) <= epsi ), break, end
    y = H(1:m,1:m)\s(1:m);
    % update approximation
    x = x + V(:,1:m)*y;
    % update approximation
    % compute residual
    r = b - A*x;
    s(j+1) = norm(r);
    error(itt+1) = s(j+1) / normb;        % check convergence
    if ( error(itt+1) <= epsi )
        break;
    end
    iter = iter+1;
end
iter = itt;
```

Matrix define in worksheet 1

Compare difference method:

Method	Number of iterations	Error	Time
GS Lower	42	3.1723e-08	0.012816
GS Upper	44	5.0821e-09	0.012816
Steepest Descent	1001	0.40714	0.053916
SOR	25	4.1048e-08	0.013954
CG	136	4.2446e-12	0.013785
GMRES	102	4.411e-14	0.045432

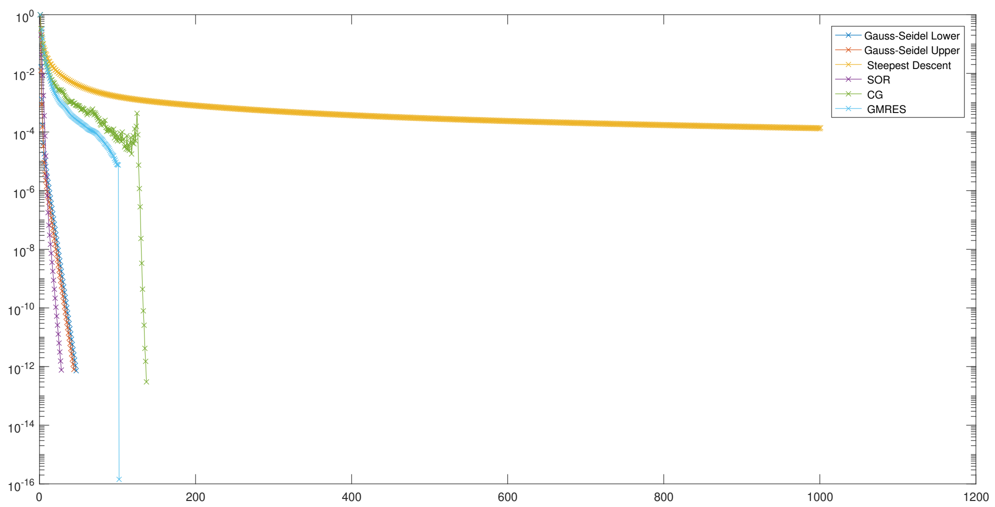

Figure 1: Comparison of value of $\|x_{aprox} - x_{exact}\|$ of different method with *semilogy*

Matrix define Laplace 1D

Compare difference method:

Method	Number of iterations	Error	Time
GS Lower	1001	4.8297	0.022161
GS Upper	1001	4.841	0.022161
Steepest Descent	1001	4.8702	0.070996
SOR	1001	4.8091	0.046463
CG	405	2.4533e-09	0.029979
GMRES	102	2.4673e-10	0.064383

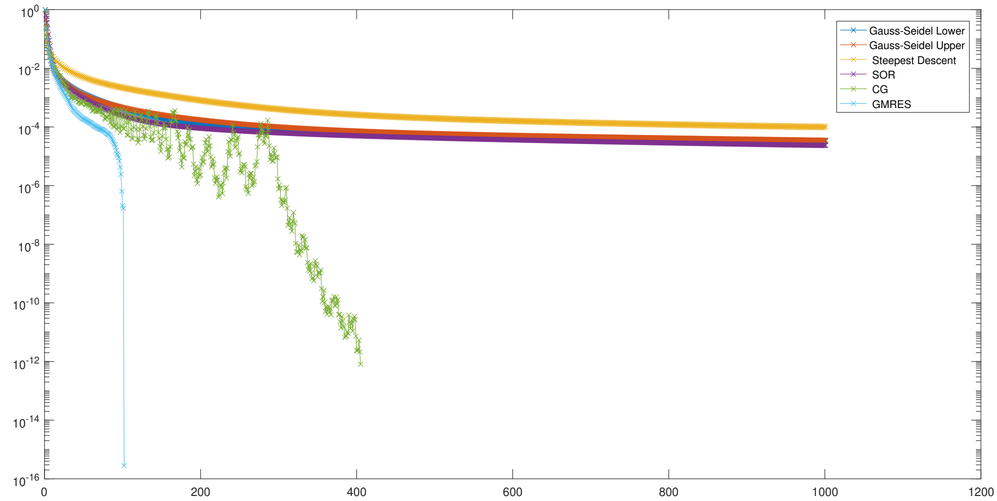

Figure 2: Comparison of value of $\|x_{aprox} - x_{exact}\|$ of different method with *semilogy*

Using matrix from Matrix Market

1 **FILE : 'hor131.mtx'**

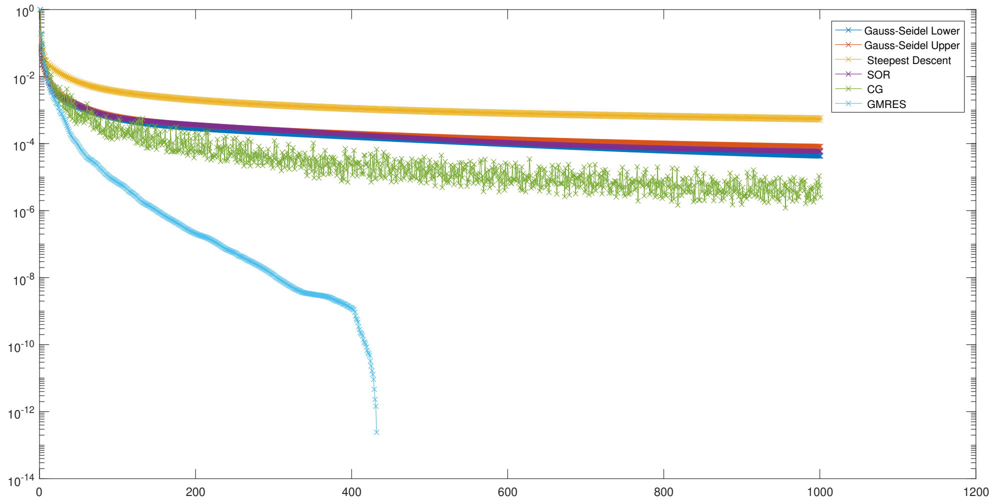

Figure 3: Comparison of value of $\|x_{aprox} - x_{exact}\|$ of different method with *semilogy*

2 FILE : 'pde225.mtx'

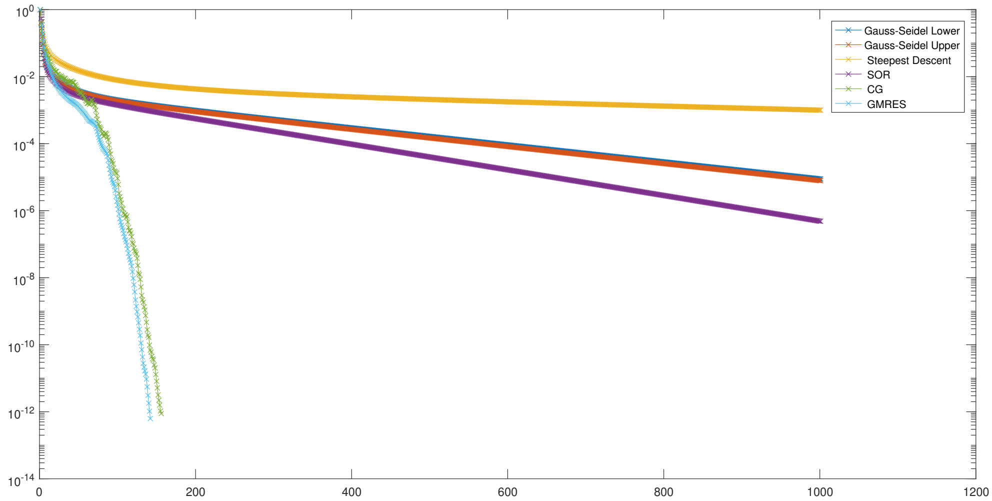

Figure 4: Comparison of value of $\|x_{aprox} - x_{exact}\|$ of different method with *semilogy*

3 FILE : 'saylr4.mtx'

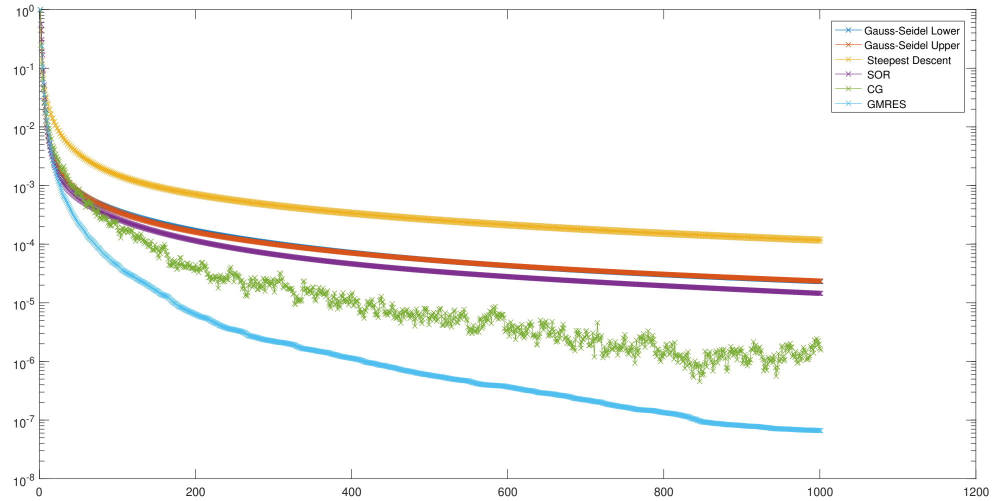

Figure 5: Comparison of value of $\|x_{aprox} - x_{exact}\|$ of different method with *semilogy*

4 FILE : 'sherman4.mtx'

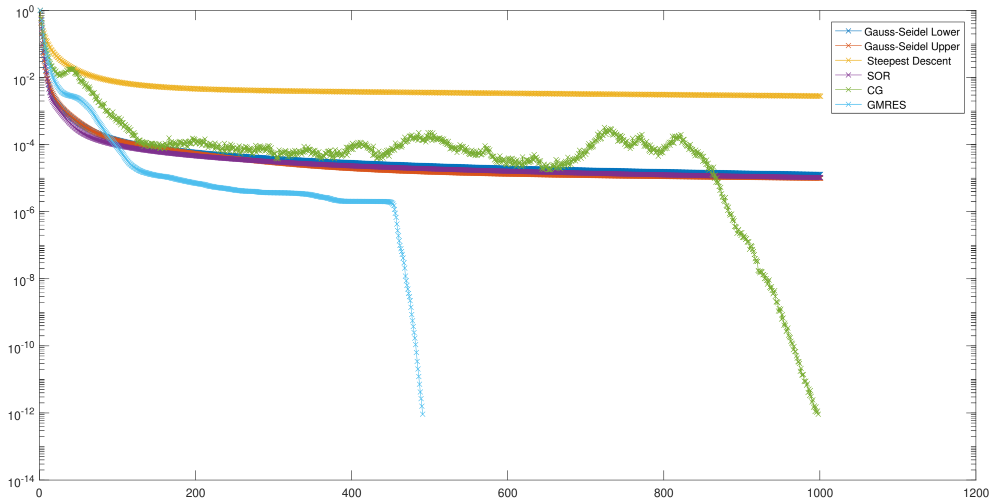

Figure 6: Comparison of value of $\|x_{aprox} - x_{exact}\|$ of different method with *semilogy*

5 FILE : 'tub1000.mtx'

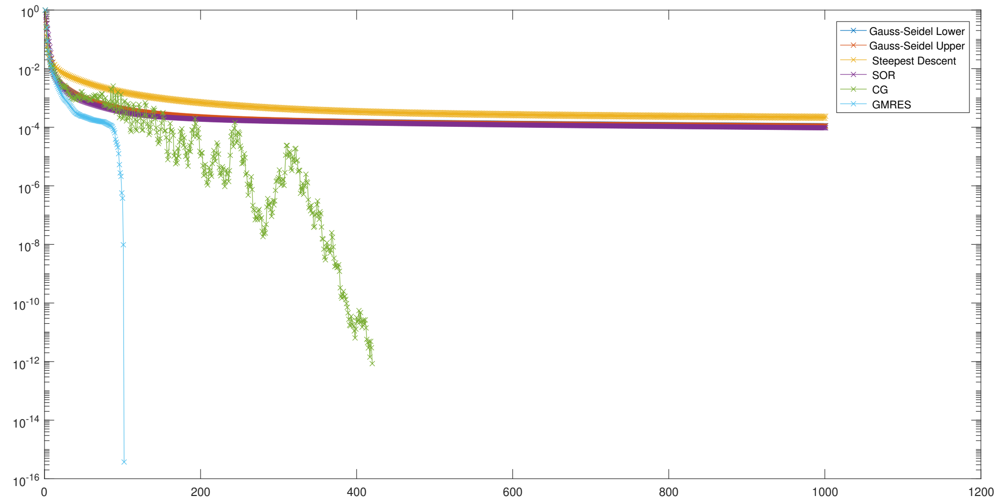

Figure 7: Comparison of value of $\|x_{aprox} - x_{exact}\|$ of different method with *semilogy*

Commend: For different case of matrix above, GMRES method show the best result among all method.

2. Exercise 2

Left preconditioner: Solve $MAx = Mb$.
Right preconditioner: Solve $AMy = b$, and then $x = My$.
Using the above **gmres.m**, modify it to **precgmres.m** so as to be able to use a right preconditioner M that can be changed at each restart of gmres.
Validate your **precgmres.m** defining $M = \text{diag}(A)$

Left preconditioner:

```
function [x, error, iter] = Left_PRECGMRES( A,b,x,M,m,itmax,epsi)
% PRECGMRES.m solves the preconditioner linear system
% The left preconditioner solve MAx = b
% The right preconditioner solve AMy = b, then x = My
% Using the above gmres.m, modify it to precgmres.m so as to be able to
   use
% a right preconditioner M that can be changed at each restart of gmres.
% input    A        REAL nonsymmetric positive definite matrix
%          x        REAL initial guess vector
%          b        REAL right hand side vector
%          M        REAL preconditioner matrix
%          numb_m   INTEGER number of iterations between restarts
%          max_it   INTEGER maximum number of iterations
%          tol      REAL error tolerance
%
% output   x        REAL solution vector
%          error    REAL error norm
%          iter     INTEGER number of iterations performed
% initialization

normb = norm(b);
```

```
if  ( normb == 0.0 )
    normb = 1.0;
end

% residual
r = M*(b - A*x);
error(1) = norm(r)/normb;
if ( error(1) < epsi )
    return;
end
[n,n] = size(A);
V(1:n,1:m+1) = zeros(n,m+1);   % Vm+1=[Vm|qm+1]
H(1:m+1,1:m) = zeros(m+1,m);   % Hessenberg matrix
cs(1:m) = zeros(m,1);
sn(1:m) = zeros(m,1);

e1    = zeros(n,1);            % basic vector
e1(1) = 1.0;
iter = 1;                      % step of iterator
itt = 1;
while iter <= itmax            % begin iteration
    r = M*(b - A*x);
    V(:,1) = r/norm(r);
    s = norm(r)*e1;
    for j = 1:m                 % construct orthonormal
        itt = itt + 1;          % basis using Gram-Schmidt
        w = M*A*V(:,j);
        for i = 1:j
            H(i,j)= w'*V(:,i);
            w = w - H(i,j)*V(:,i);
        end
        %     size(H)
        H(j+1,j) = norm(w);
        V(:,j+1) = w/H(j+1,j);
        % We tranform the Hessenberg matrix H into a triangular matrix by applying Givens rotation
        for k = 1:j-1                                   % apply Givens rotation
            temp     =  cs(k)*H(k,j) + sn(k)*H(k+1,j);
            H(k+1,j) = -sn(k)*H(k,j) + cs(k)*H(k+1,j);
            H(k,j)   = temp;
        end
        cs(j) = H(j,j)/sqrt(H(j,j)^2 + H(j+1,j)^2);
        sn(j) = H(j+1,j)/sqrt(H(j,j)^2 + H(j+1,j)^2);
        temp   = cs(j)*s(j);                             % approximate residual norm
        s(j+1) = -sn(j)*s(j);
        s(j)   = temp;
        H(j,j) = cs(j)*H(j,j) + sn(j)*H(j+1,j);
        H(j+1,j) = 0.0;
        % Which also provides the error at iteration j+1

        error(itt+1) = abs(s(j+1)) / normb;

        if ( error(itt+1) <= epsi )                        % update approximation
            y = H(1:j,1:j)\s(1:j);                      % and exit
            x = x + V(:,1:j)*y;
```

```
            % error(i+1) = abs(s(i+1)) / bnrm2;
            break;
        end
    end

    if ( error(itt+1) <= epsi ), break, end
    y = H(1:m,1:m)\s(1:m);
%         x = x + V*y;                                    % update
    approximation
    x = x + V(:,1:m)*y;                                   % update
    approximation
    % compute residual
    r = M*(b - A*x);
    s(j+1) = norm(r);
    error(itt+1) = s(j+1) / normb;                          % check
    convergence
    if ( error(itt+1) <= epsi )
        break;
    end
    iter = iter+1;
end
iter = itt;
```

Right preconditioner:

```
function [x, error, iter] = Right_PRECGMRES( A,b,y,M,m,itmax,epsi)
% PRECGMRES.m solves the preconditioner linear system
% The left preconditioner solve MAx = b
% The right preconditioner solve AMy = b, then x = My
% Using the above gmres.m, modify it to precgmres.m so as to be able to
  use
% a right preconditioner M that can be changed at each restart of gmres.
% input    A         REAL nonsymmetric positive definite matrix
%          x         REAL initial guess vector
%          b         REAL right hand side vector
%          M         REAL preconditioner matrix
%          numb_m    INTEGER number of iterations between restarts
%          max_it    INTEGER maximum number of iterations
%          tol       REAL error tolerance
%
% output   x         REAL solution vector
%          error     REAL error norm
%          iter      INTEGER number of iterations performed
% initialization

normb = norm(b);
if  ( normb == 0.0 )
    normb = 1.0;
end

% residual
r = b - A*M*y;
error(1) = norm(r)/normb;
if ( error(1) < epsi )
    return;
end
[n,n] = size(A);
V(1:n,1:m+1) = zeros(n,m+1);    % Vm+1=[Vm|qm+1]
H(1:m+1,1:m) = zeros(m+1,m);    % Hessenberg matrix
```

```
cs(1:m) = zeros(m,1);
sn(1:m) = zeros(m,1);

e1    = zeros(n,1);             % basic vector
e1(1) = 1.0;
iter = 1;                       % step of iterator
itt = 1;
while iter <= itmax             % begin iteration
    r = b - A*M*y;
    V(:,1) = r/norm(r);
    s = norm(r)*e1;
    for j = 1:m                  % construct orthonormal
        itt = itt + 1;           % basis using Gram-Schmidt
        w = A*M*V(:,j);
        for i = 1:j
            H(i,j)= w'*V(:,i);
            w = w - H(i,j)*V(:,i);
        end
        %      size(H)
        H(j+1,j) = norm(w);
        V(:,j+1) = w/H(j+1,j);
        % We tranform the Hessenberg matrix H into a triangular matrix by applying Givens rotation
        for k = 1:j-1                              % apply Givens rotation
            temp     =  cs(k)*H(k,j) + sn(k)*H(k+1,j);
            H(k+1,j) = -sn(k)*H(k,j) + cs(k)*H(k+1,j);
            H(k,j)   = temp;
        end
        cs(j) = H(j,j)/sqrt(H(j,j)^2 + H(j+1,j)^2);
        sn(j) = H(j+1,j)/sqrt(H(j,j)^2 + H(j+1,j)^2);
        temp   = cs(j)*s(j);                        % approximate residual norm
        s(j+1) = -sn(j)*s(j);
        s(j)   = temp;
        H(j,j) = cs(j)*H(j,j) + sn(j)*H(j+1,j);
        H(j+1,j) = 0.0;
        % Which also provides the error at iteration j+1

        error(itt+1) = abs(s(j+1)) / normb;

        if ( error(itt+1) <= epsi )                    % update approximation
            z = H(1:j,1:j)\s(1:j);                  % and exit
            y = y + V(:,1:j)*z;
            % error(i+1) = abs(s(i+1)) / bnrm2;
            break;
        end
    end

    if ( error(itt+1) <= epsi ), break, end
    z = H(1:m,1:m)\s(1:m);
%        x = x + V*y;                                   % update approximation
    y = y + V(:,1:m)*z;                                % update approximation
    % compute residual
    r = b - A*M*y;
```

```
    s(j+1) = norm(r);
    error(itt+1) = s(j+1) / normb;                              % check
   convergence
    if ( error(itt+1) <= epsi )
        break;
    end
    iter = iter+1;
end
iter = itt;
x = M*y;
```

We using **right preconditioner** to apply equation $Ax = b$ become $AMy = b$ and $x = My$.

Where $M = \mathrm{diag}(\mathrm{diag}(A))$

Method	error	iter	Time (seconds)
GMRES	1.006e-14	101	0.047792
Right preconditioner	2.3357e-13	101	0.060019

Table 1: Compare 2 method method

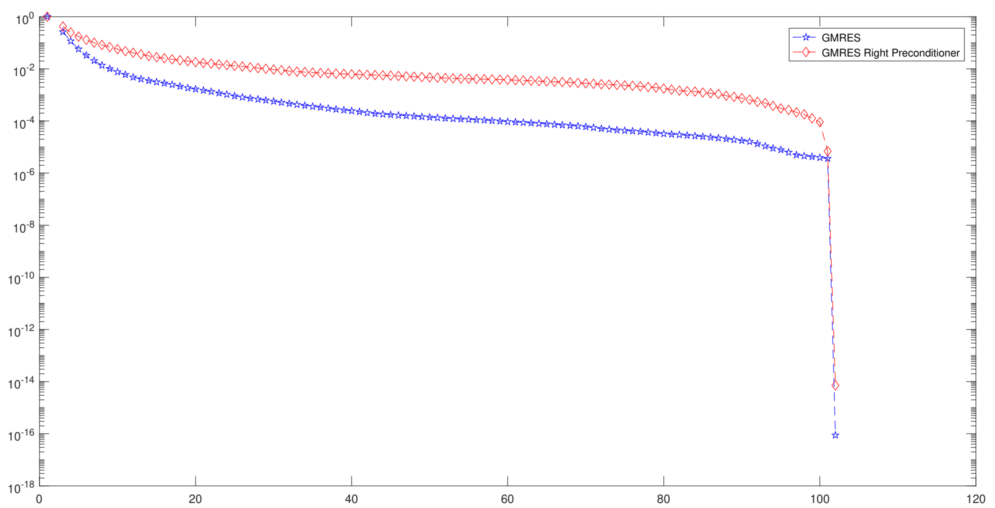

Figure 8: Comparison of GMRES and PRECGMRES

Commend: Although GMRES convergence faster, but PRECGMRES show us a better result (the approx is more close to exact solution)

3. Exercise 3

Look into the matlab function **ILU** that performs an ILU(0) approximation of A. [L0, U0] = ILU(A) Use this ILU(0) as a right preconditioner in precgmres.m to solve Ax = b as in TP1

Answer

Using **ILU** function, we have:

$$[L, U] = \text{ilu}(A)$$
$$M = \text{inv}(L * U)$$

Matrix in worksheet 1:

Method	error	iter	Time (seconds)
GMRES	2.4282e-14	101	0.040265
Right preconditioner	8.4696e-15	2	0.03

Table 2: Compare 2 method method

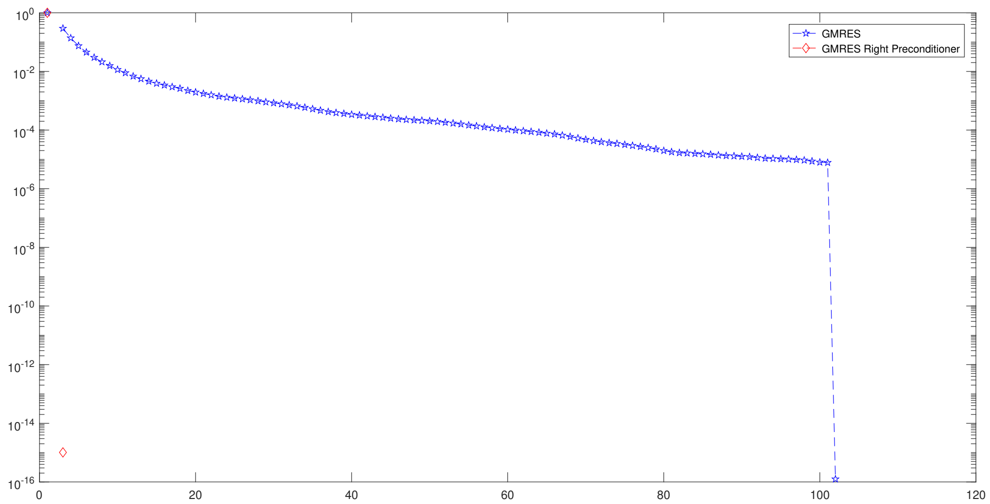

Figure 9: Comparison of GMRES and PRECGMRES

Matrix Market
File 'hor131.mtx'

Method	error	iter	Time (seconds)
GMRES	3.2943e-06	427	0.35305
Right preconditioner	2.4151e-07	44	0.59493

Table 3: Compare 2 method method

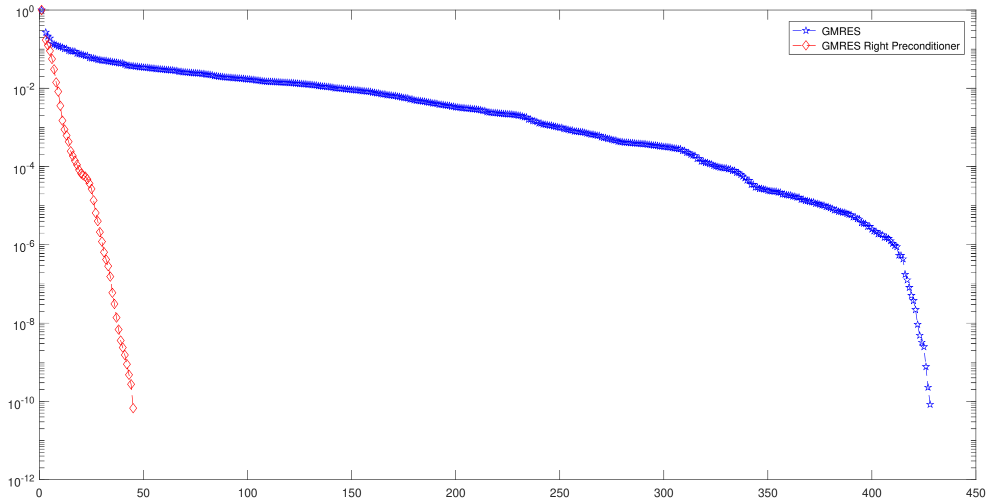

Figure 10: Comparison of GMRES and PRECGMRES

File 'pde225.mtx'

Method	error	iter	Time (seconds)
GMRES	1.0984e-09	76	0.033946
Right preconditioner	7.9933e-10	21	0.068774

Table 4: Compare 2 method method

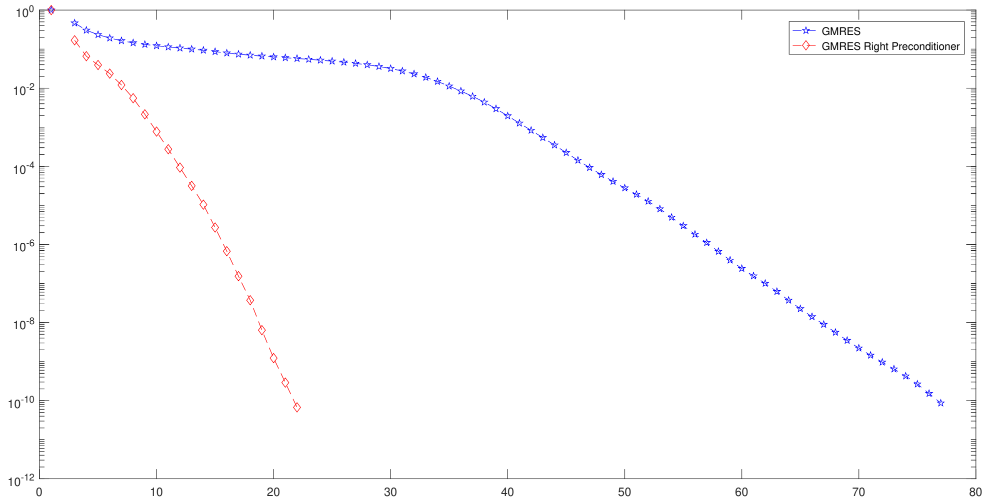

Figure 11: Comparison of GMRES and PRECGMRES

File 'saylr4.mtx'

Method	error	iter	Time (seconds)
GMRES	2.5793e-05	1684	41.2416
Right preconditioner	8.3633e-05	52	38.0274

Table 5: Compare 2 method method

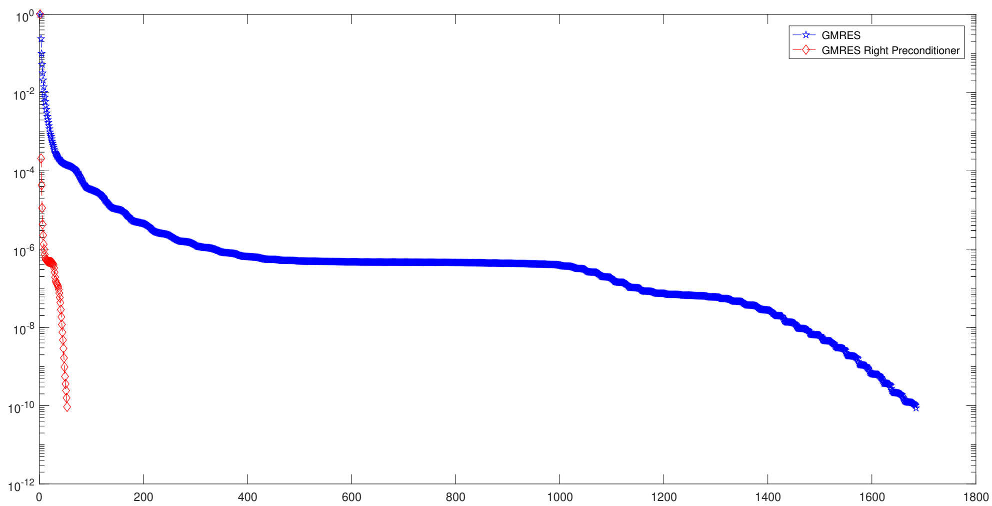

Figure 12: Comparison of GMRES and PRECGMRES

File 'sherman4.mtx'

Method	error	iter	Time (seconds)
GMRES	7.8963e-09	137	0.11605
Right preconditioner	4.3628e-09	38	0.61389

Table 6: Compare 2 method method

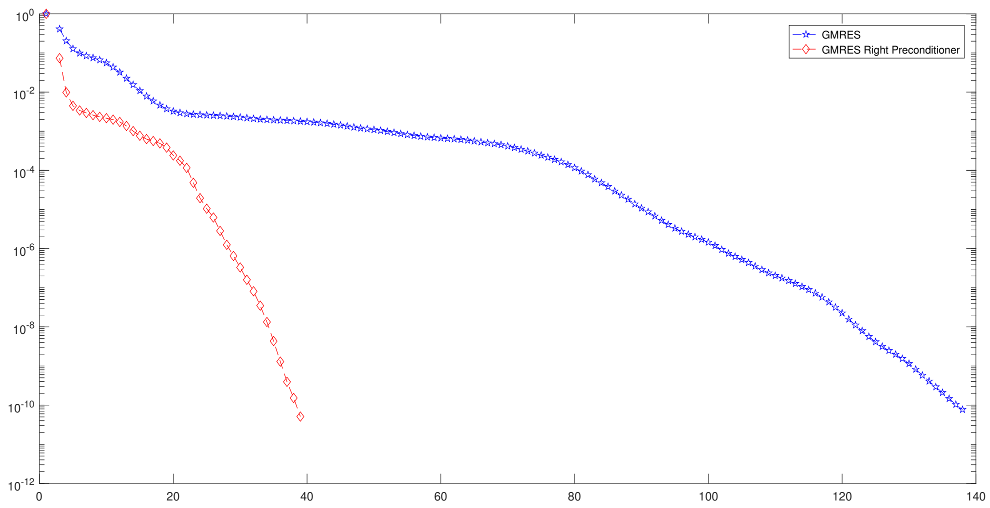

Figure 13: Comparison of GMRES and PRECGMRES

File 'tub100.mtx'

Method	error	iter	Time (seconds)
GMRES	4.4087e-13	101	0.046172
Right preconditioner	3.4059e-11	17	0.035072

Table 7: Compare 2 method method

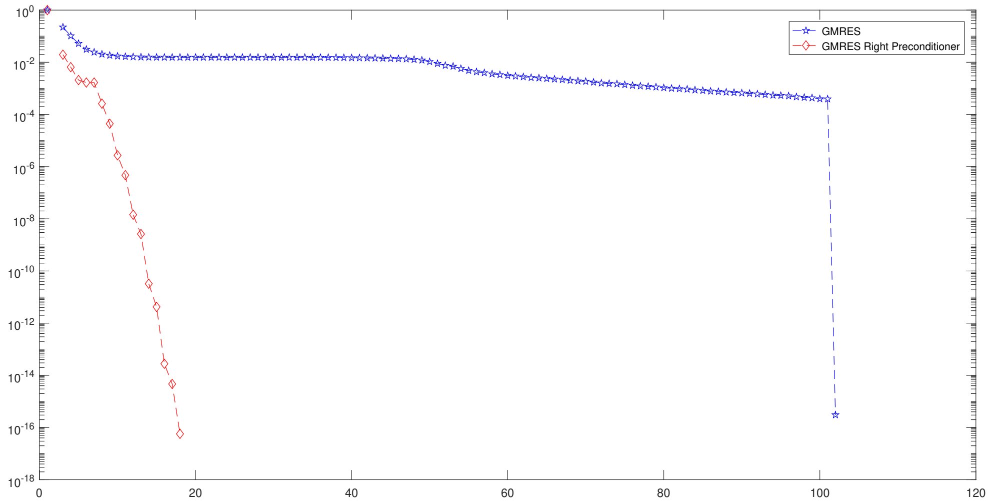

Figure 14: Comparison of GMRES and PRECGMRES