3)

Here we are to solve the equation

$$\begin{bmatrix} \frac{1}{1} & \frac{1}{2^1} & \frac{1}{2^2} & \frac{1}{2^3} & \frac{1}{2^4} \\ \frac{1}{1} & \frac{1}{4^1} & \frac{1}{4^2} & \frac{1}{4^3} & \frac{1}{4^4} \\ \frac{1}{1} & \frac{1}{6^1} & \frac{1}{6^2} & \frac{1}{6^3} & \frac{1}{6^4} \\ \frac{1}{1} & \frac{1}{8^1} & \frac{1}{8^2} & \frac{1}{8^3} & \frac{1}{8^4} \\ \frac{1}{1} & \frac{1}{10^1} & \frac{1}{10^2} & \frac{1}{10^3} & \frac{1}{10^4} \end{bmatrix} \begin{pmatrix} \epsilon_\infty \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} -2 \\ -1.5 \\ -1.4343 \\ -1.4128 \\ -1.4031 \end{pmatrix}$$

Firstly we apply the SOR method as the instruction said, however it seems that the algorithm always diverges.

To demonstrate the divergence, we firstly calculate the solution by built-in matrix-division algorithm, then we add a small deviation to that solution, and set it as our initial guess. It comes out it will always diverge.

Results: % i is the iteration time

```
x0=A\b+0.00001*rand(5,1)

x0 =

  -1.3875

   0.0361

  -2.0396

   1.7267

  -5.3840

>> [x1,i]=GaussSeidelRelaxInt(A,b,x0,0.00001,1,100)

ans =diagonally dominant condition not satisfied

x1 =

  1.0e+232 *

  -0.0012

   0.0044

   0.0155

   0.1961

   3.8738

i =

  100
```

Code for SOR

```
function [x1,i]=GaussSeidelRelaxInt(A,b,x0,tol,w,n)
% solve the linear equation Ax=b by Gauss-Seidel-Relaxation method:
A=D(diagonal)+L(left)+R(right)
% (D+L)x<i+1>=b-Rx<i> with every update of x is multiplied by factor
w
% sufficient condition : diagonally dominant
% necessary condition : all absolute values of eigenvalues of A is
smaller than 1
% x0 : initial guess (column vector!)
% tol : tolerance
% w : factor w
% n : largest iteration times

T=sum(abs(A),2)-abs(diag(A))*2;
if max(T) < 0
    'diagonally dominant condition satisfied'
else
    'diagonally dominant condition not satisfied'
end

LD=A;
[m,~]=size(A);
for i=1:(m-1)
    LD(i,(i+1):end)=0;
end
R=A-LD;

T=b-R*x0;
x1=zeros(m,1);
x1(1)=(T(1)/LD(1,1)-x0(1))*w+x0(1);
for j=2:m
    x1(j)=(( T(j)- LD(j,1:(j-1))*x1(1:(j-1)) )/LD(j,j) -
x0(j))*w+x0(j);
end
i=1;

while ( (norm((x1-x0)./x0)) > tol ) && ( i<n )
    x0=x1;
    T=b-R*x0;
    x1(1)=(T(1)/LD(1,1)-x0(1))*w+x0(1);
    for j=2:m
      x1(j)=(( T(j)- LD(j,1:(j-1))*x1(1:(j-1)) )/LD(j,j) -
x0(j))*w+x0(j);
    end
    i=i+1;
end
```

and we apply the Power method to find the dominant eigenvalue of matrix A, to check if it meets the necessary condition for convergence, i.e., smaller than 1(absolute value).

```
e_val =

    1.6677
```

the dominant eigenvalue comes out to be 1.6677, larger than 1, so SOR method won't work here.

Then we apply the Gauss Elimination method, LU decomposition method.

Gauss method doesn’t work here, it comes out the result deviate badly from the result of build-in matrix-division x0.

```
>> x1=GaussPivotScale(A,b)

x1 =

   1.0e+04 *

   -0.0002

    0.0006

   -0.0052

    0.0353

    2.8735

>> x1-x0

ans =

   1.0e+04 *

   -0.0001

    0.0006

   -0.0050

    0.0351

    2.8741
```

The LU decomposition method works well, since the L and U matrix are not too much ill-conditioned. And the result only deviates a little from the build-in matrix-division result.

```
   >> [L,U]=deLU(A)

   L =

       1.0000         0         0         0         0

       1.0000    1.0000         0         0         0

       1.0000    1.3333    1.0000         0         0

       1.0000    1.5000    1.6875    1.0000         0

       1.0000    1.6000    2.1600    2.0480    1.0000

   U =

       1.0000    0.5000    0.2500    0.1250    0.0625

            0   -0.2500   -0.1875   -0.1094   -0.0586

            0         0    0.0278    0.0255    0.0164

            0         0         0   -0.0020   -0.0020
```

```
        0         0         0         0    0.0001

>> x=ForwardSub(L,b)

x =

   -2.0000

    0.5000

   -0.1010

    0.0076

   -0.0005

>> x=BackSub(U,x)

x =

   -1.3875

    0.0361

   -2.0396

    1.7267

   -5.3840

>> x-x0

ans =

   1.0e-12 *

   -0.0020

    0.0402

   -0.2838

    0.8133

   -0.7816
```

Code for Gauss elimination

```
function x = GaussPivotScale(A,b)
%
% GaussPivotScale uses Gauss elimination with partial
% pivoting and row scaling to solve the linear system
% Ax = b.
%
% x = GaussPivotScale(A,b) where
%
% A is the n-by-n coefficient matrix,
% b is the n-by-1 result vector,
%
% x is the n-by-1 solution vector.
n = length(b);
A = [A b]; % Augmented matrix
```

```
for k = 1:n-1,
M = max(abs(A(k:end, k:end-1)), [], 2);
% Find maximum for each row
a = abs(A(k:end, k)); % Find maximum for kth column
[~,I] = max(a./M);
% Find relative row with maximum ratio
I = I + k-1; % Adjust relative row to actual row
if I > k
    % Pivot rows
A([k I],:) = A([I k],:);
end
m = A(k+1:n, k)/A(k, k); % Construct multipliers
[Ak,M] = meshgrid(A(k,:), m); % Create mesh
A(k+1:n,:) = A(k+1:n,:) - Ak.*M;
end
Ab = A;
% Find the solution vector using back substitution
x = BackSub(A,b);
```

Code for LU decomposition

```
function [L,U]=deLU(A)
% calculate the LU decomposition of matrix A
% here A is a square matrix
[n,~]=size(A);
L=eye(n);
U=zeros(n);
for i=1:n
    U(i,i:n)=A(i,i:n);
    L((i+1):n,i)=A((i+1):n,i)/U(i,i);
    A=A-L(1:n,i)*U(i,1:n);
end
```