Computational Problem 2

Table of Contents

Setup

```
close all, clear all, clc
format long, format compact
fs = 16;
set(0,'defaulttextfontsize',fs);
set(0,'defaultaxesfontsize',fs);
```

EXERCISE 1

(a)

Since matrix A_ϵ is a pentadiagonal matrix, so only main diagonal, and the first two upper and two lower diagonals exists nonzero elements.

Thus, for size n, each row has a maximum of 5 nonzero elements. Because if A_ϵ is strictly diagonal dominant, it must satisfy $|a_{ii}| > \sum_{j=1,\ j\neq i}^{n} |a_{ij}|$, for $i = 1, \ldots, n$.

So, matrix A_ϵ is strictly diagonal dominant if $1 > 2 \times (\epsilon^2 + \epsilon)$. And we can get $\epsilon < 0.366$.

Therefore, $\epsilon \in [0, 0.366)$.

```
syms e
cond1 = e >= 0;
cond2 = e <= 1;
cond3 = 2*(e^2 + e) < 1;
conds = [cond1 cond2 cond3];

sol = solve(conds, e, 'ReturnConditions', true);
vpa(sol.conditions)
```

```
ans =
0.0 <= x & x < 0.36602540378443864676372317075294
```

(b)

```
tol = 1e-10;
nmax = 1000;
x0 = [0; 0; 0; 0; 0];
[A, b] = matrix(5, 0.3);

% Jacobi method
[x, niter, relresiter, xiter] = itermeth(A, b, x0, nmax, tol, 'J');
niter

% Gauss-Seidel method
[x, niter, relresiter, xiter] = itermeth(A,b,x0,nmax,tol,'G');
niter
```

```
itermeth converged in 50 iterations.
niter =
    50
itermeth converged in 14 iterations.
niter =
    14
```

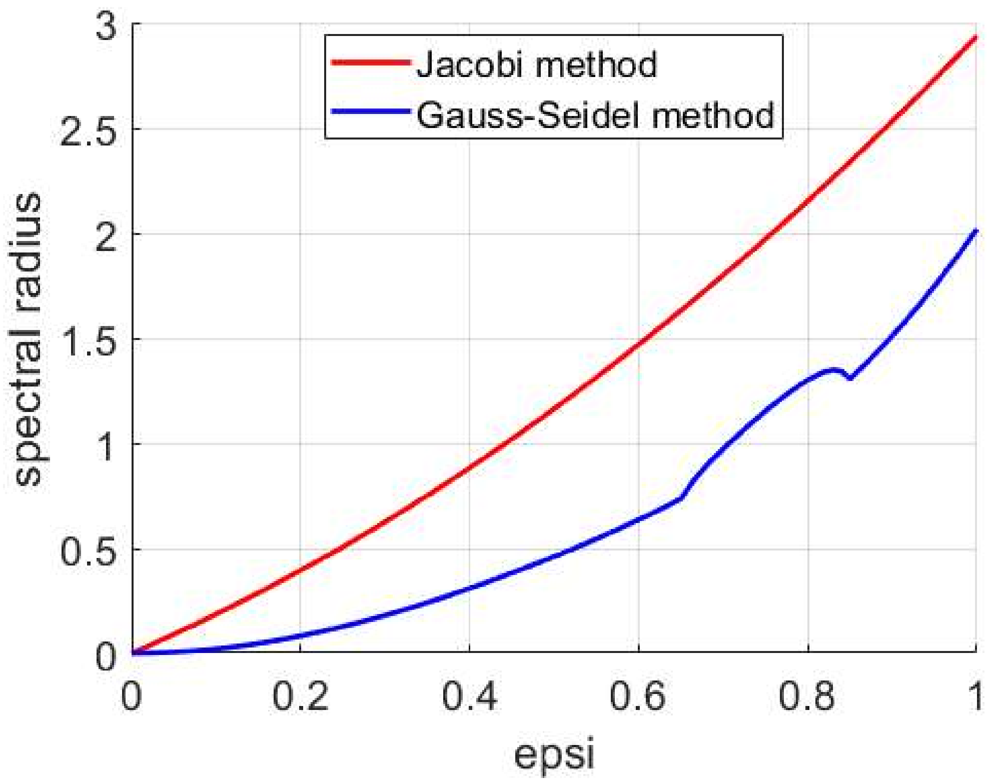
For Jacobi method, it needs 50 iterations to convergence; and for Gauss-Seidel method, it requires 14 iterations to convergence.

(c)

Plot the spectral radius against the value of ϵ for both methods

```
x=linspace(0, 1, 101);
figure
grid on
xlabel('epsi')
ylabel('spectral radius')
hold on

radius_BJ = zeros(size(101));
radius_BGS = zeros(size(101));

for e = 0:0.01:1
    [A, b] = matrix(5, e);
    D = diag(diag(A));  % diagonal part of A_epsi
    L = tril(A,-1);  % lower triangular part of A_epsi
    U = triu(A,1);  % lower triangular part of A_epsi

    B_J = -(D^(-1)) * (L + U);
    B_GS = -(D + L)^(-1) * U;

    radius_BJ(round(e*100) + 1) = max(abs(eig(B_J)));  % spectral radius is
 the maximum modulus of eigenvalues
    radius_BGS(round(e*100) + 1) = max(abs(eig(B_GS)));
end
```

```
plot(x, radius_BJ, 'r','LineWidth', 2);
plot(x, radius_BGS, 'blue','LineWidth', 2);
legend({'Jacobi method', 'Gauss-Seidel method'}, 'Location', 'Best');
```



Since the Jacobi method and the Gauss-Seidel method converge if the spectral radius of both B_J and B_{GS} less than 1. Therefore, according to the graph, when $x \in [0, 0.44]$, both B_J and B_{GS} less than 1, and this result is 0.074 larger than the answer of 0.366 from question (a).

According to the graph, if both methods converge, the Gauss-Seidel method is faster than the Jacobi method, because its spectral radius is smaller than the Jacobi method.

When n = 5 and ϵ = 0.5, I recommend using the Gauss-Seidel method.

```
[A, b] = matrix(5, 0.5);
D = diag(diag(A));  % diagonal part of A_epsi
L = tril(A,-1);  % lower triangular part of A_epsi
U = triu(A,1);  % lower triangular part of A_epsi

B_J = -(D^(-1)) * (L + U);
B_GS = -(D + L)^(-1) * U;

radius_BJ = max(abs(eig(B_J)))  % spectral radius is the maximum modulus of
 eigenvalues
radius_BGS = max(abs(eig(B_GS)))

tol = 1e-10;
nmax = 1000;
```

```
x0 = [0; 0; 0; 0; 0];
[A, b] = matrix(5, 0.5);

% Jacobi method
[x, niter, relresiter, xiter] = itermeth(A, b, x0, nmax, tol, 'J');

% Gauss-Seidel method
[x, niter, relresiter, xiter] = itermeth(A,b,x0,nmax,tol,'G');
```

```
radius_BJ =
   1.163568373244976
radius_BGS =
   0.460186129658958
itermeth reached the maximum iteration number without converging.
itermeth converged in 27 iterations.
```

Since spectral radius of B_J larger than 1, the Jacobi method will not convergence. Therefore, it should use the Gauss-Seidel method When n = 5 and $\epsilon = 0.5$.

EXERCISE 2

(a)

```
list_N = [5, 10, 20, 40, 80];
rhoBJ = zeros(size(length(list_N)));
rhoBGS = zeros(size(length(list_N)));

for i = 1:length(list_N)
    h = 1 / list_N(i);
    A = (2/h^2)*diag(ones(list_N(i)-1,1)) - (1/
h^2)*diag(ones(list_N(i)-2,1),1) - (1/h^2)*diag(ones(list_N(i)-2,1),-1);
    b = transpose(sin(pi*h*(1:list_N(i)-1)));

    D = diag(diag(A));
    L = tril(A) - D;
    U = triu(A) - D;

    B_J = -(D^(-1)) * (L + U);
    B_GS = -(D + L)^(-1) * U;

    rhoBJ(i) = max(abs(eig(B_J)));  % spectral radius is the maximum modulus
 of eigenvalues
    rhoBGS(i) = max(abs(eig(B_GS)));
end

rhoBJ, rhoBGS
```

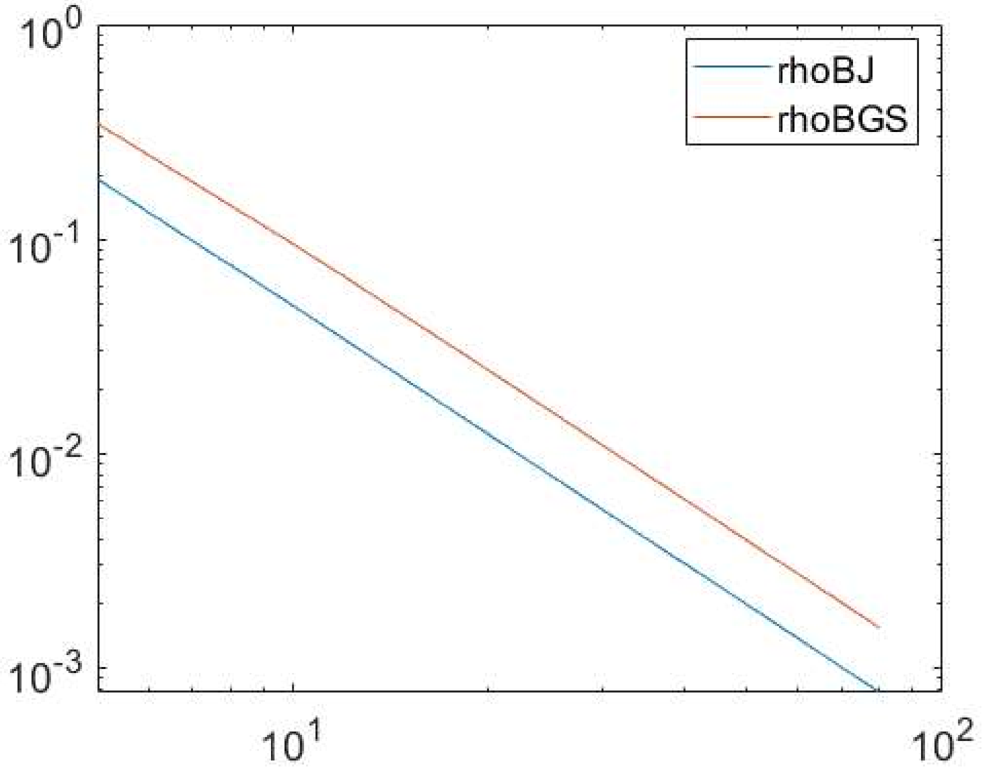
```
rhoBJ =
  Columns 1 through 3
   0.809016994374948    0.951056516295154    0.987688340595138
  Columns 4 through 5
   0.996917333733128    0.999229036240723
rhoBGS =
  Columns 1 through 3
   0.654508497187474   0.904508497187474   0.975528258147580
  Columns 4 through 5
   0.993844170297569   0.998458666866563
```

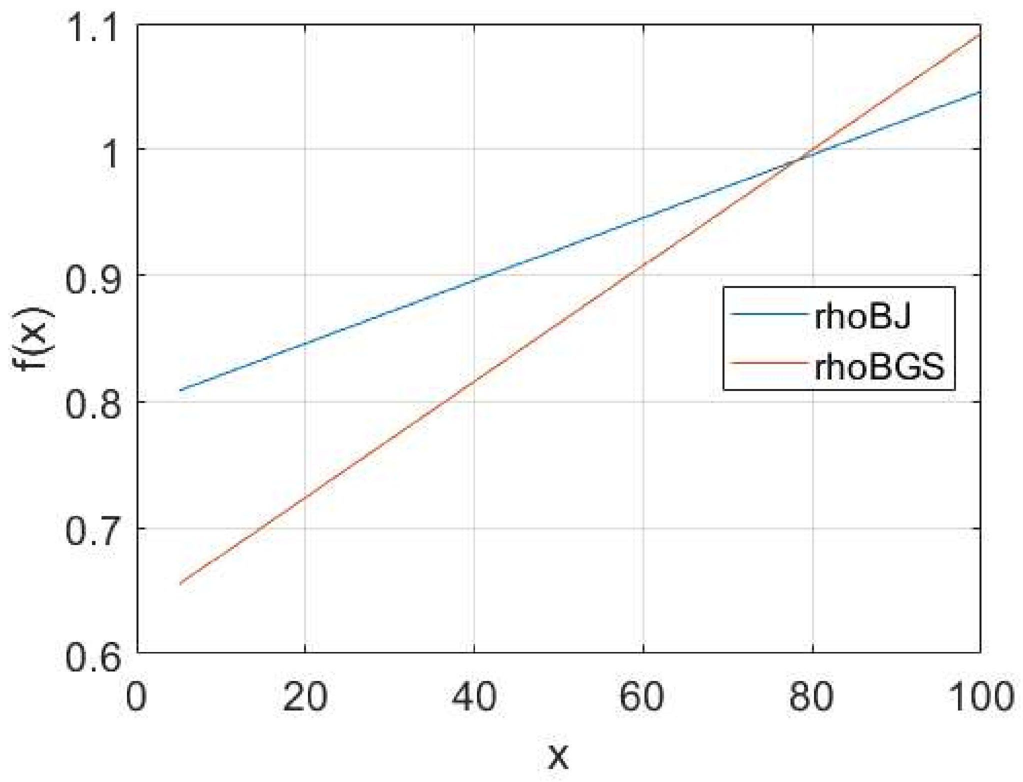
The two iterative methods will convergent for these values of N, because each of spectral radius less than 1. And the Gauss-Seidel method converge faster, because every value in rhoBGS less than rhoBJ.

As size N increases, both $\rho(B_J)$ and $\rho(B_{GS})$ are grow up. Thus, I expect that the performance of the Jacobi and Gauss-Seidel methods will decline with the size N continues to increase.

```
figure

Nvec = [5, 10, 20, 40, 80];
loglog(Nvec, 1 - rhoBJ);
hold;
loglog(Nvec, 1 - rhoBGS);

legend({'rhoBJ', 'rhoBGS'}, 'Location', 'Best');


% get relationship between size N and spectral radius

% for $\rho B_J$
syms a b;
f1 = a*list_N(1) + b == 1 - rhoBJ(1);
f2 = a*list_N(5) + b == 1 - rhoBJ(5);
ab_J = solve(f1, f2, a, b);
vpa(ab_J.a), vpa(ab_J.b)

% for $\rho B_{GS}$
syms a b;
f1 = a*list_N(1) + b == 1 - rhoBGS(1);
f2 = a*list_N(5) + b == 1 - rhoBGS(5);
ab_GS = solve(f1, f2, a, b);
vpa(ab_GS.a), vpa(ab_GS.b)

f_J = @(x) 0.0025 * x + 0.796;
f_GS = @(x) 0.0046 * x + 0.632;

% relationship between size N and spectral radius for both methods

x=linspace(5, 100, 100);               % Define a set of x values for plotting
figure                              % Create a new figure
plot(x, f_J(x))           % Plot f
hold
plot(x, f_GS(x))           % Plot f

legend({'rhoBJ', 'rhoBGS'}, 'Location', 'Best');
grid on
xlabel('x')
ylabel('f(x)')

Current plot held
```

```
ans =
-0.0025361605582103363687451746955048
ans =
0.2036638084161041195585539753665 2
ans =
-0.0045860022623878649028483778238297
ans =
0.3684215141244657099051096338 3713
Current plot held
```





According to the graph above, we can observe that the function between size N and (1 - spectral radius) is a straight line. So, we can deduce that their relationship is linear, and the function is: (1 - sr) = aN + b, where sr is spectral radius, a and b are parameters.

For $\rho(B_{\mathrm{J}})$, we can calculate that a = -0.0025, b = 0.204, so the final relationship is sr = 0.0025N - 0.204 + 1 => sr = 0.0025N + 0.796.

For $\rho(B_{\mathrm{GS}})$, we can calculate that a = -0.0046, b = 0.368, so the final relationship is sr = 0.0046N - 0.368 + 1 => sr = 0.0046N + 0.632.

According to the functions above, as size N increases, the spectral radius also increases, so that the number of iterations to achieve a fixed solution accuracy will increases as well. Additionally, the spectral radius may greater than 1 when N approaches infinity, so the two methods will not converge, and the iterations number will also approaches infinity.

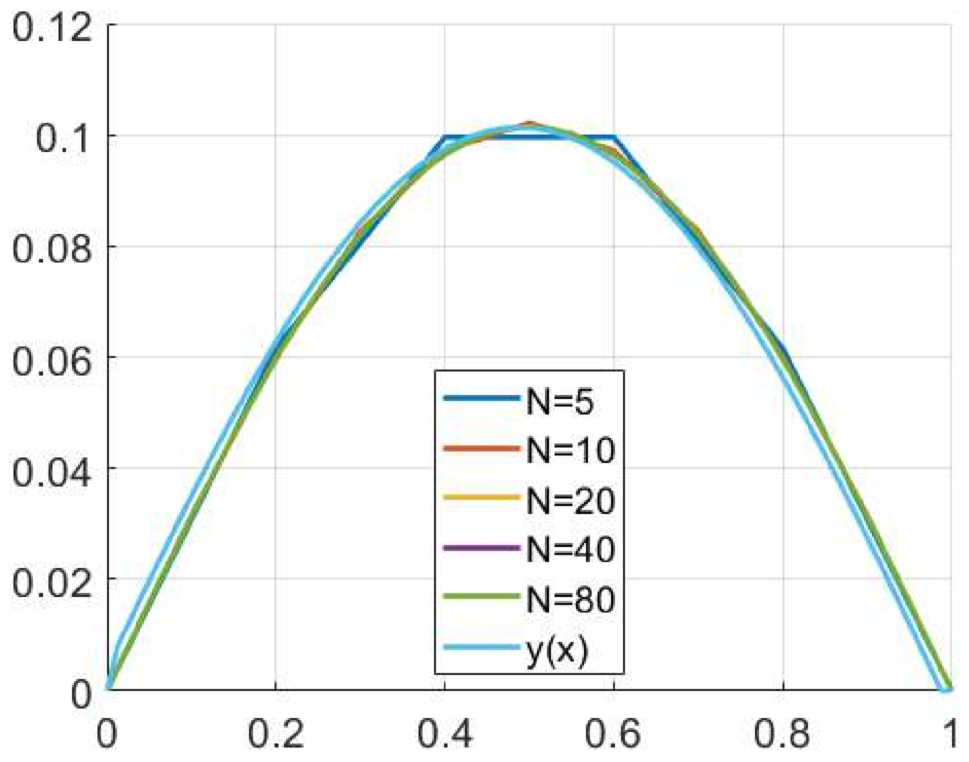
The cost of a direct method is generally O(n^3), where n is the size of matrix; and for iterative method, the cost of each iteration in general is O(n^2). So, the Jacobi and Gauss-Seidel methods cheaper than direct method if the number of iterations to be much less than N. However, when N is large, iteration number will larger than N if we use Jacobi and Gauss-Seidel methods to solve this problem. Therefore, in terms of practicality, using the Jacobi and Gauss-Seidel methods for this problem is inferior to using direct method.

(b)

```
figure
grid on
hold on
```

```
tol = 1e-10;
nmax = 10^5;

list_N = [5, 10, 20, 40, 80];

% Plot resulting solutions of Gauss-Seidel method

for i = 1:length(list_N)
    h = 1 / list_N(i);
    A = (2/h^2)*diag(ones(list_N(i)-1,1)) - (1/
h^2)*diag(ones(list_N(i)-2,1),1) - (1/h^2)*diag(ones(list_N(i)-2,1),-1);
    b = transpose(sin(pi*h*(1:list_N(i)-1)));
    x0 = transpose(zeros(1, list_N(i)-1));

    [x, niter, relresiter, xiter] = itermeth(A, b, x0, nmax, tol, 'G');

    result = zeros(1, list_N(i) + 1);
    result(2:list_N(i)) = x;
    range = linspace(0, 1, list_N(i) + 1);

    plot(range, result,'LineWidth', 2);
end

% Plot exact solution y(x) on the finest mesh (N = 80)
N = 80;
h = 1 / N;
y = @(x) pi^(-2) * sin(pi * x);
range = linspace(0, 1, N + 1);
result = zeros(1, N + 1);

for i = 2:N
    result(i) = y(i * h);
end

plot(range, result,'LineWidth', 2);

legend({'N=5', 'N=10', 'N=20', 'N=40', 'N=80', 'y(x)'}, 'Location', 'Best');
```

```
itermeth converged in 56 iterations.
itermeth converged in 231 iterations.
itermeth converged in 931 iterations.
itermeth converged in 3731 iterations.
itermeth converged in 14929 iterations.
```

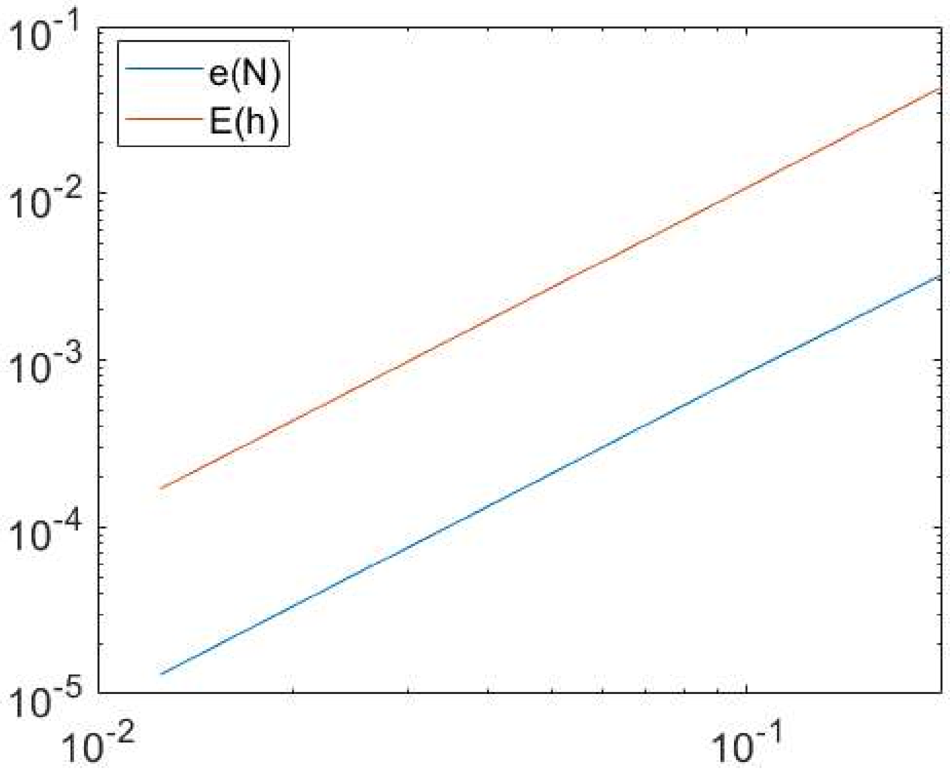


According to the graph, as N increases, the solution appears closer to the line of exact solution, and the error also becomes smaller. Thus, the solutions appear to converge as N increases.

```
list_N = [5, 10, 20, 40, 80];
error_vect = zeros(1, length(list_N));

y = @(x) pi^(-2) * sin(pi * x);

for i = 1:length(list_N)
    h = 1 / list_N(i);
    A = (2/h^2)*diag(ones(list_N(i)-1,1)) - (1/
h^2)*diag(ones(list_N(i)-2,1),1) - (1/h^2)*diag(ones(list_N(i)-2,1),-1);
    b = transpose(sin(pi*h*(1:list_N(i)-1)));
    x0 = transpose(zeros(1, list_N(i)-1));

    [x, niter, relresiter, xiter] = itermeth(A, b, x0, nmax, tol, 'G');

    % calculate y(x_n)
    y_n = transpose(zeros(1, list_N(i)-1));
    for j = 1:list_N(i)-1
        y_n(j) = y(j * h);
    end

    error_vect(i) = max(abs(x - y_n));

end
```

```
error_vect

figure
hvect = 1./list_N;
loglog(hvect, error_vect)  % e(N)
hold


syms c
cond1 = c * hvect(1)^2 >= error_vect(1);
cond2 = c * hvect(5)^2 >= error_vect(5);
conds = [cond1 cond2];

C = solve(conds, c);
vpa(C)

loglog(hvect, C*hvect.^2);  % E(h)

legend({'e(N)', 'E(h)'}, 'Location', 'Best');
```

```
itermeth converged in 56 iterations.
itermeth converged in 231 iterations.
itermeth converged in 931 iterations.
itermeth converged in 3731 iterations.
itermeth converged in 14929 iterations.
error_vect =
  Columns 1 through 3
   0.003233759448575   0.000837461821339   0.000208590596337
  Columns 4 through 5
   0.000052099391046   0.000013021827269
Current plot held
ans =
1.0833396945217224072166573023424
```

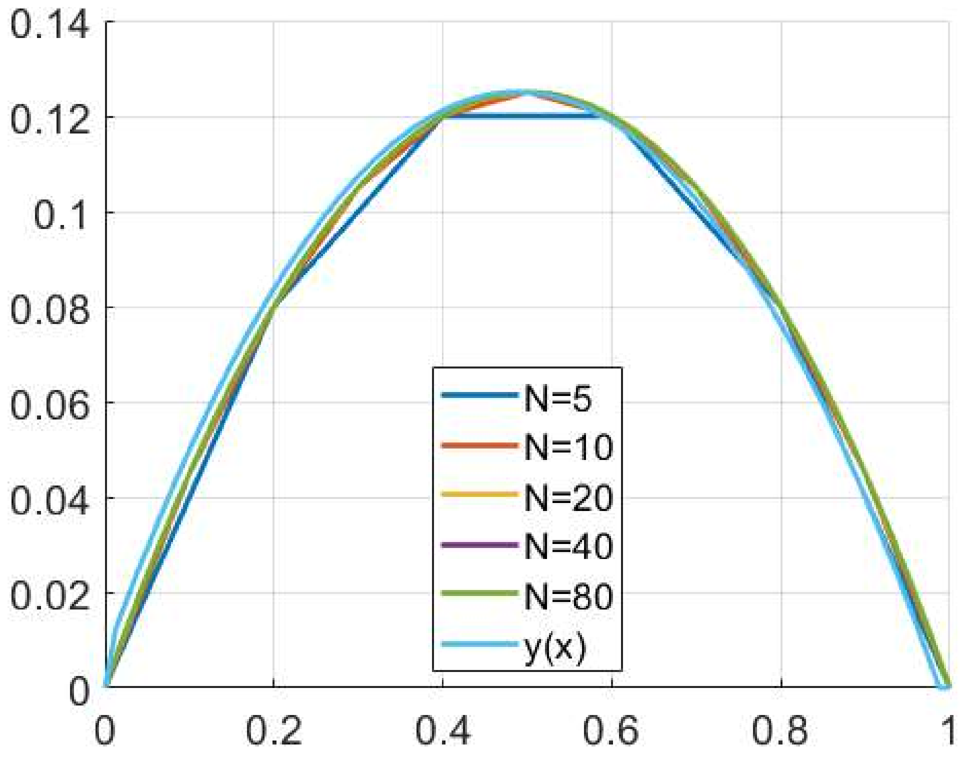


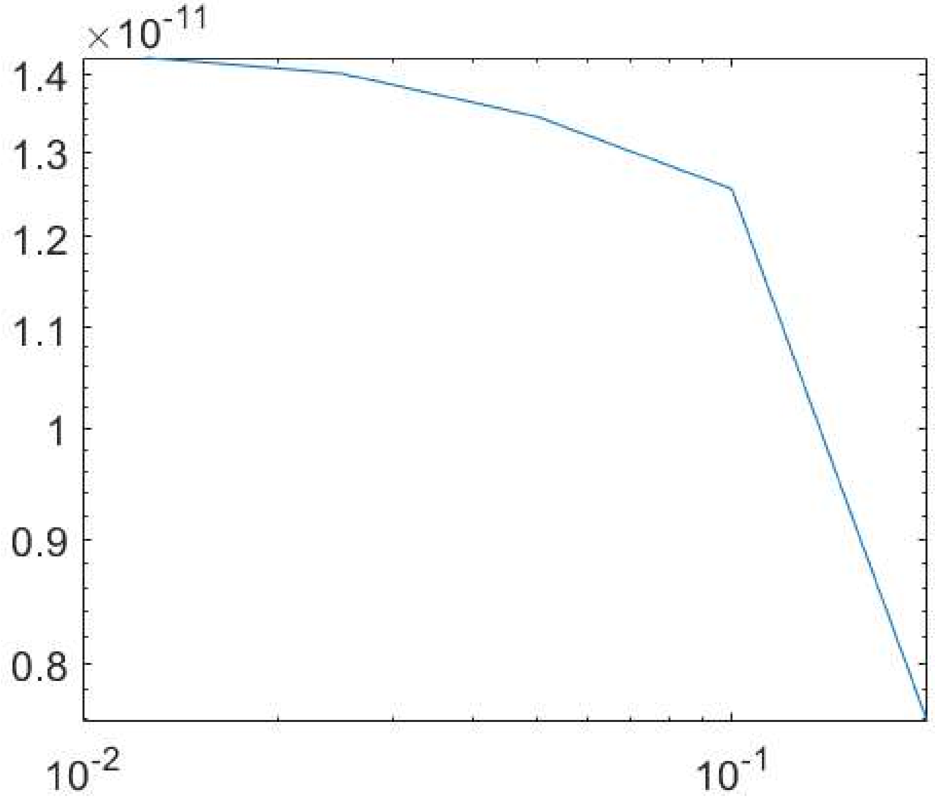
According to the assumption of question, $e(N) \leq Ch^2$ as h = 1/N $\rightarrow$ 0, so we can know that error e(N) is proportional to the step size h = 1/N from the graph. To verify the assumption, we set p=2, and find C by the graph. Thus, C=1.0833396945217224072166573023424. Finally, we can see that the line of e(N) lower than the line of E(h), which means $e(N) \leq Ch^2$. Therefore, the numerical results support this theoretical estimate.

(c)

```
figure
grid on
hold on

tol = 1e-10;
nmax = 10^5;

list_N = [5, 10, 20, 40, 80];

% Plot resulting solutions of Gauss-Seidel method

for i = 1:length(list_N)
    h = 1 / list_N(i);
    A = (2/h^2)*diag(ones(list_N(i)-1,1)) - (1/
h^2)*diag(ones(list_N(i)-2,1),1) - (1/h^2)*diag(ones(list_N(i)-2,1),-1);
    b = transpose(ones(1, list_N(i)-1));  % modify
    x0 = transpose(zeros(1, list_N(i)-1));
    [x, niter, relresiter, xiter] = itermeth(A, b, x0, nmax, tol, 'G');

    result = zeros(1, list_N(i) + 1);
    result(2:list_N(i)) = x;
    range = linspace(0, 1, list_N(i) + 1);

    plot(range, result,'LineWidth', 2);
end

% Plot exact solution y(x) on the finest mesh (N = 80)
N = 80;
h = 1 / N;
y = @(x) (1 / 2) * x * (1 - x);
range = linspace(0, 1, N + 1);
result = zeros(1, N + 1);

for i = 2:N
    result(i) = y(i * h);
end

plot(range, result,'LineWidth', 2);

legend({'N=5', 'N=10', 'N=20', 'N=40', 'N=80', 'y(x)'}, 'Location', 'Best');


% Corresponding errors

Nvec = [5, 10, 20, 40, 80];
error_vect = zeros(1, length(Nvec));

y = @(x) (1 / 2) * x * (1 - x);

for i = 1:length(Nvec)
    h = 1 / Nvec(i);
    A = (2/h^2)*diag(ones(Nvec(i)-1,1)) - (1/h^2)*diag(ones(Nvec(i)-2,1),1) -
 (1/h^2)*diag(ones(Nvec(i)-2,1),-1);
    b = transpose(ones(1, Nvec(i)-1));  % modify
    x0 = transpose(zeros(1, Nvec(i)-1));

    [x, niter, relresiter, xiter] = itermeth(A, b, x0, nmax, tol, 'G');

    % calculate y(x_n)
    y_n = transpose(zeros(1, Nvec(i)-1));
    for j = 1:Nvec(i)-1
        y_n(j) = y(j * h);
    end

    error_vect(i) = max(abs(x - y_n));

end

figure
hvect = 1./Nvec;
loglog(hvect, error_vect)
```

```
itermeth converged in 56 iterations.
itermeth converged in 230 iterations.
itermeth converged in 928 iterations.
itermeth converged in 3716 iterations.
itermeth converged in 14865 iterations.
itermeth converged in 56 iterations.
itermeth converged in 230 iterations.
itermeth converged in 928 iterations.
itermeth converged in 3716 iterations.
itermeth converged in 14865 iterations.
```





For this case, there is no longer a relationship of $e(N) \leq Ch^2$ as h = 1/N $\rightarrow$ 0. Since we replaced $sin(\pi x)$ with 1, there is no x exists in d^2/dx^2, so constant C is not proportional to $\max_{x\in[0,1]} |y''''(x)|$.

Published with MATLAB® R2022b