```
%Conjugate gradients

function u = CG(A, b, tol)

n = size(A,1);

% Intial guesss uo
uo = zeros(n,1);

ro = b - A*uo;
po = ro;


for k = 1:10000
    wo = A*po;
    alphao = (ro'*ro)/(po'*wo);
    uk = uo + alphao*po;
    rk = ro - alphao*wo;

    if norm(rk,2)<tol*norm(b,2)
        break;
    end

    betao = (rk'*rk)/(ro'*ro);
    pk = rk + betao*po;

    uo = uk;
    ro = rk;
    po = pk;
end

fprintf('%3d %12.4e\n', k, norm(rk));

u = uk;

end
```

```
Not enough input arguments.

Error in CG (line 5)
n = size(A,1);
```

Published with MATLAB® R2020a

Contents


```
% Script for testing fd2poisson over the square [a,b]x[a,b]
a1 = 0; b1 = 1;
m = (2^7)-1;
h = (b1-a1)/(m+1);

[x,y] = meshgrid(a1:h:b1); %Uniform mesh, including boundary points.

% Laplacian(u) = f
f = @(x,y) 10*pi^2*(1+cos(4*pi*(x+2*y))-2*sin(2*pi*(x+2*y))).*exp(sin(2*pi*(x+2*y)));
% u = g on Boundary
g = @(x,y) exp(sin(2*pi*(x+2*y)));

% Exact solution is g.
uexact = @(x,y) g(x,y);

tol = 10^(-8);

idx = 2:m+1;
idy = 2:m+1;

% Compute boundary terms, south, north, east, west
ubs = feval(g,x(1,1:m+2),y(1,1:m+2));     % Include corners
ubn = feval(g,x(m+2,1:m+2),y(m+2,1:m+2)); % Include corners
ube = feval(g,x(idy,m+2),y(idy,m+2));     % No corners
ubw = feval(g,x(idy,1),y(idy,1));         % No corners

% Evaluate the RHS of Poisson's equation at the interior points.
f1 = feval(f,x(idy,idx),y(idy,idx));

% Adjust f for boundary terms
f1(:,1) = f1(:,1) - ubw/h^2;             % West
f1(:,m) = f1(:,m) - ube/h^2;             % East
f1(1,1:m) = f1(1,1:m) - ubs(idx)/h^2;    % South
f1(m,1:m) = f1(m,1:m) - ubn(idx)/h^2;    % North

b = reshape(f1,m*m,1);

I = eye(m);
ze = zeros(m,1);
e = ones(m,1);
T1 = spdiags([ze -2*e ze],[-1 0 1],m,m);
S1 = spdiags([e e],[-1 1],m,m);
T2 = spdiags([e -2*e e],[-1 0 1],m,m);
S2 = spdiags([ze ze],[-1 1],m,m);

D2x = (1/h^2)*(kron(I, T1) + kron(S1,I));
D2y = (1/h^2)*(kron(I, T2) + kron(S2,I));

A = D2x +D2y;


%Using the conjugate gradient.
u = CG(A,b,tol);

%Reshape u for plotting
u = reshape(u,m,m);

% Append on to u the boundary values from the Dirichlet condition.
u = [ubs;[ubw,u,ube];ubn];
```

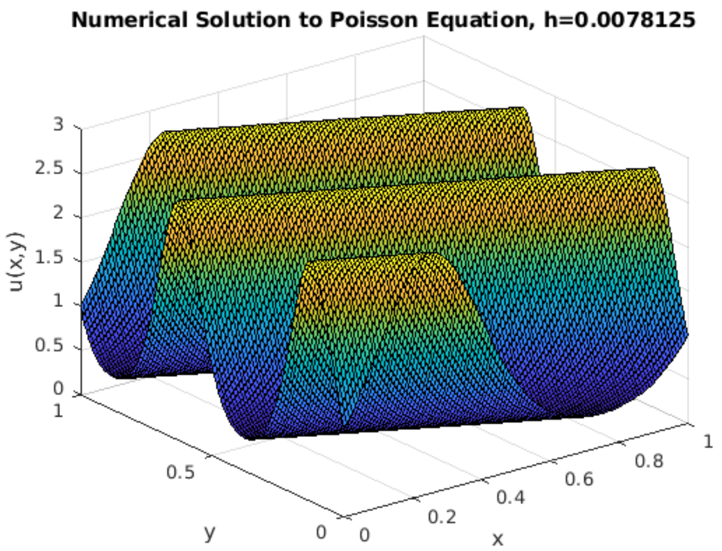
```
346   5.4475e-03
```

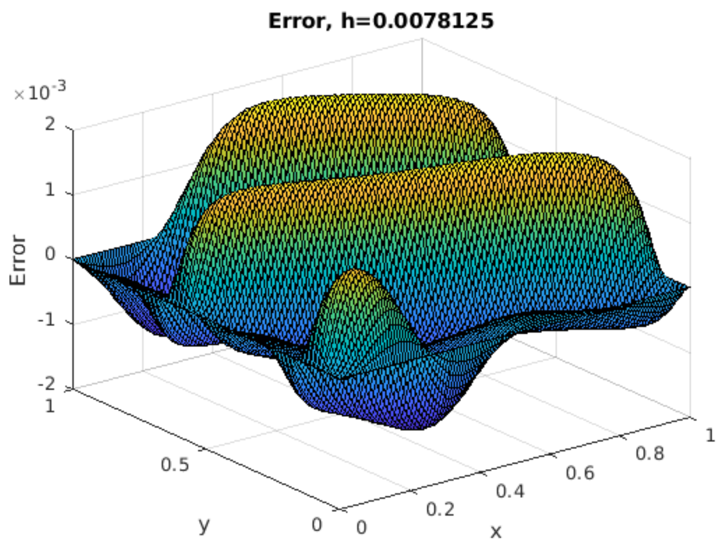
Plot solution

```
figure, set(gcf,'DefaultAxesFontSize',10,'PaperPosition', [0 0 3.5 3.5]),
surf(x,y,u), xlabel('x'), ylabel('y'), zlabel('u(x,y)'),
title(strcat('Numerical Solution to Poisson Equation, h=',num2str(h)));
%Plot error
figure, set(gcf,'DefaultAxesFontSize',10,'PaperPosition', [0 0 3.5 3.5]),
surf(x,y,u-uexact(x,y)),xlabel('x'),ylabel('y'), zlabel('Error'),
title(strcat('Error, h=',num2str(h)));

t2   = zeros(1,3);
t1   = [];
t4   = zeros(1,3);
t3   = [];

for ii = 1:3
    %Using the conjugate gradient.
    u = CG(A,b,tol);

    %Reshape u for plotting
    u = reshape(u,m,m);
```

```
    % Append on to u the boundary values from the Dirichlet condition.
    tic
    u = [ubs;[ubw,u,ube];ubn];
    gedirect = toc;
    t2(ii) = gedirect;

    h = (b1-a1)/(m+1);
    w = 2/(1+sin(pi*h)); %optimal relaxation parameter
    tic
    [usor,x,y] = fd2poissonsor(f,g,a1,b1,m,w);
    gedirect = toc;
    t4(ii) = gedirect;

end
t1   = [t1,t2];
t3   = [t3,t4];
fprintf('The number of iterations the CG code takes to converge, k = 346 with norm(rk) = 5.4475e-03 in an average time of %d\n',mean(t1));
fprintf('SOR method takes an average time of %d\n',mean(t3))
fprintf('Comparing the timing between CG and SOR, its seen that CG converges in a short time faster than SOR method.\n')
```

```
346   5.4475e-03
346   5.4475e-03
346   5.4475e-03
The number of iterations the CG code takes to converge, k = 346 with norm(rk) = 5.4475e-03 in an average time of 5.683333e-04
SOR method takes an average time of 2.355237e-01
Comparing the timing between CG and SOR, its seen that CG converges in a short time faster than SOR method.
```





Published with MATLAB® R2020a

No. 2 Cell-Centered Poisson Solver

a) $u''(y) = f(y)$, $0 \leq y \leq 1$, with $u(0) = \alpha_1$, $u(1) = \alpha_2$

$y_k = (k - \frac{1}{2})h$, $k = 1, \ldots, m$, $h = 1/m$

at $k = 2, 3, \ldots, m-1$

$$U_k'' = \frac{U_{k-1} - 2U_k + U_{k+1}}{h^2} = f_k \quad \text{--- } ①$$

at $k = 1$

$$U_1'' = \frac{U_0 - 2U_1 + U_2}{h^2} = f_1$$

$$U_0 - 2U_1 + U_2 = h^2 f_1 \quad \text{--- } ②$$

at $u(0) = \alpha_1$

$$\frac{U_0 - 2u(0) + U_1}{(h/2)^2} = f(0)$$

$$U_0 = 2\alpha_1 - U_1 + \frac{h^2}{4} f(0) \quad \text{--- } ③$$

Putting ③ into ②

$$2\alpha_1 - U_1 + \frac{h^2}{4} f(0) - 2U_1 + U_2 = h^2 f_1$$

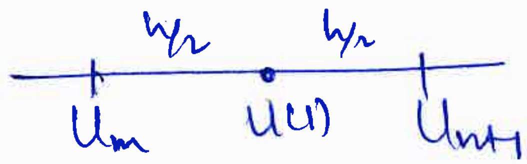
$$U_2 - 3U_1 = -2\alpha_1 + \frac{h^2}{4}(4f_1 - f(0))$$

at $k = m$, equation ① becomes

$$U_m'' = \frac{U_{m-1} - 2U_m + U_{m+1}}{h^2} = f_m \quad \text{--- } ④$$

$$U_{m-1} - 2U_m + U_{m+1} = h^2 f_m$$

at $u(1) = \alpha_2$



$$U_m - 2U(1) + U_{m+1} = \frac{h^2}{4} f(1)$$

$$U_{m+1} = 2U(1) - U_m + \frac{h^2}{4} f(1) \quad \text{———} \quad (5)$$

Put (5) into (4)

$$U_{m-1} - 3U_m = -2\alpha_2 + \frac{h^2}{4}\left(4f_m - f(1)\right)$$

Therefore the second-order accurate approximation will be:

$$\begin{cases} U_2 - 3U_1 = -2\alpha_1 + \dfrac{h^2}{4}\left(4f_1 - 4f(0)\right), & k=1 \\ U_{m-1} - 3U_m = -2\alpha_2 + \dfrac{h^2}{4}\left(4f_m - f(1)\right), & k=m \\ U_k'' = \dfrac{U_{k-1} - 2U_k + U_{k+1}}{h^2} = f_k, & k = 2,3,4,\dots,m \end{cases}$$

b) $u''(x) = f(x)$, $0 \le x \le 1$ with $u'(0) = \beta_1$, $u'(1) = \beta_2$

$x_k = (k - \frac{1}{2})h$, $k = 1,\dots,m$, $h = 1/m$.

at x_k

$$U_k'' = \frac{U_{k-1} - 2U_k + U_{k+1}}{h^2} = f_k, \quad k = 2,\dots,m-1.$$

at $k=1$

$$U_1'' = \frac{U_0 - 2U_1 + U_2}{h^2} = f_1$$

$U_0 - 2U_1 + U_2 = h^2 f_1$ ——— (*)

Using fictitious point at x_0

$$U'(0) = \frac{U_0 - U_1}{h} = \beta_1 \Rightarrow U_0 = h\beta_1 + U_1$$

Substituting U_0 into (*)

$$h\beta_1 + U_1 - 2U_1 + U_2 = h^2 f_1$$

$$U_2 - U_1 = h^2 f_1 - h\beta_1$$

at $k = m$

$$U''_m = \frac{U_{m-1} - 2U_m + U_{m+1}}{h^2} = f_m$$

$$U_{m-1} - 2U_m + U_{m+1} = h^2 f_m \quad --- (**)$$

Using fictitious point at x_{m+1}

$$U'(1) = \frac{U_{m+1} - U_m}{h} = \beta_2 \Rightarrow U_{m+1} = U_m + h\beta_2 \quad ---- (***)$$

Substitute U_{m+1} into (**) we obtain

$$U_{m-1} - U_m = h^2 f_m - \beta_2 h$$

therefore the second order accurate approximation becomes:

$$\begin{cases} U_2 - U_1 = h^2 f_1 - h\beta_1 , & k = 1 \\ U''_k \neq U_{k-1} - 2U_k + U_{k+1} = h^2 f_k , & k = 2, \dots, m-1 \\ U_{m-1} - U_m = h^2 f_m - \beta_2 h ; & k = m \end{cases}$$

2(C)

from (a)

~~D_{2y}~~

$$D_{2y} = \begin{bmatrix} -3 & 1 & & & & 0 \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & 1 & -2 & 1 \\ & & & 1 & -2 & 1 \\ & & & & 1 & -3 \end{bmatrix}$$

from (b)

$$D_{2x} = \begin{bmatrix} -1 & 1 & & & 0 \\ 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & \ddots & \ddots & \ddots \\ 0 & & 1 & -2 & 1 \\ & & & 1 & -1 \end{bmatrix}$$

Combining (a) and (b), we have

$$\left[D_{2y} \otimes I + I \otimes D_{2x} \right] U = f \quad \text{—} ①$$

where I is an identity matrix of dimension equal to that of D_{2y} and D_{2x}.

U is the solution we are looking for

f is a vector

Equation ① is second order approximated

No. 3 Linear multistep methods

Complete the missing entries of this table.

Case	Char. eq	Roots	Stability	accuracy	Consistency	local truncation error	Convergence
a)	$w^2-\frac{1}{2}w-\frac{1}{2}=0$	$1, -\frac{1}{2}$	Yes	0	No	$-\frac{1}{2}u'(t_n)$	No
b)	$w-1=0$	1	Yes	0	No	$u'(t_n)$	No
c)	$w^4-1=0$	$\pm 1, \pm i$	Yes	2	Yes	$\frac{4}{3}k^2 u^{(3)}(t_n)$	Yes
d)	$w^3-w=0$	$\pm 1, 0$	Yes	3	Yes	$\frac{1}{3}k^3 u^{(4)}(t_n)$	Yes
e)	$w^4-\frac{8}{19}w^3+\frac{8}{19}w-1=0$	$\pm 1, \frac{4}{19}\pm\frac{\sqrt{345}}{19}i$	No	6	Yes	$\frac{-6}{665}k^6 u^{(7)}(t_n)$	No
f)	$w^3+w^2-w-1=0$	± 1	Yes	2	Yes	$\frac{2}{3}k^2 u^{(3)}(t_n)$	Yes

The Corresponding stencil

a)

	u	f
t_{n+1}	$\square$	
t_n	$\alpha_1=-\frac{1}{2}$	$\beta_1=2$
t_{n-1}	$\alpha_0=-\frac{1}{2}$	

b)

	u	f
t_{n+1}	$\square$	
t_n	$\alpha_0=-1$	

c)

	u	f
t_{n+1}		
t_n		$\beta_3 = 4/3$
t_{n-1}		$\beta_2 = 4/3$
t_{n-2}		$\beta_1 = 4/3$
t_{n-3}	$\alpha_0 = -1$	

d)

	u	f
t_{n+1}		
t_n		$\beta_2 = 7/3$
t_{n-1}	$\alpha_1 = -1$	$\beta_1 = -2/3$
t_{n-2}		$\beta_0 = 1/3$

e)

	u	f
t_{n+1}		$\beta_4 = 6/19$
t_n	$\alpha_3 = -8/19$	$\beta_3 = 24/19$
t_{n-1}		
t_{n-2}	$\alpha_1 = 8/19$	$\beta_1 = 24/19$
t_{n-3}	$\alpha_0 = -1$	$\beta_0 = 6/19$

f)

	u	f
t_{n+1}		
t_n	$\alpha_2 = 1$	$\beta_2 = 2$
t_{n-1}	$\alpha_1 = -1$	$\beta_1 = 2$
t_{n-2}	$\alpha_0 = -1$	

State the Generating polynomials $\rho(w)$ and $\sigma(w)$

a) $\rho(w) = w^2 - \frac{1}{2}w - \frac{1}{2}$

$\sigma(w) = 2w$

b) $\rho(w) = w - 1$

$\sigma(w) = 0$

c) $\rho(w) = w^4 - 1$

$\sigma(w) = \frac{4}{3}w^3 + \frac{4}{3}w^2 + \frac{4}{3}w$

d) $\rho(w) = w^3 - w$

$\sigma(w) = \frac{7}{3}w^2 - \frac{2}{3}w + \frac{1}{3}$

e) $\rho(w) = w^4 - \frac{8}{19}w^3 + \frac{8}{19}w - 1$

$\sigma(w) = \frac{6}{19}w^4 + \frac{24}{19}w^3 + \frac{24}{19}w + \frac{6}{19}$

f) $\rho(w) = w^3 + w^2 - w - 1$

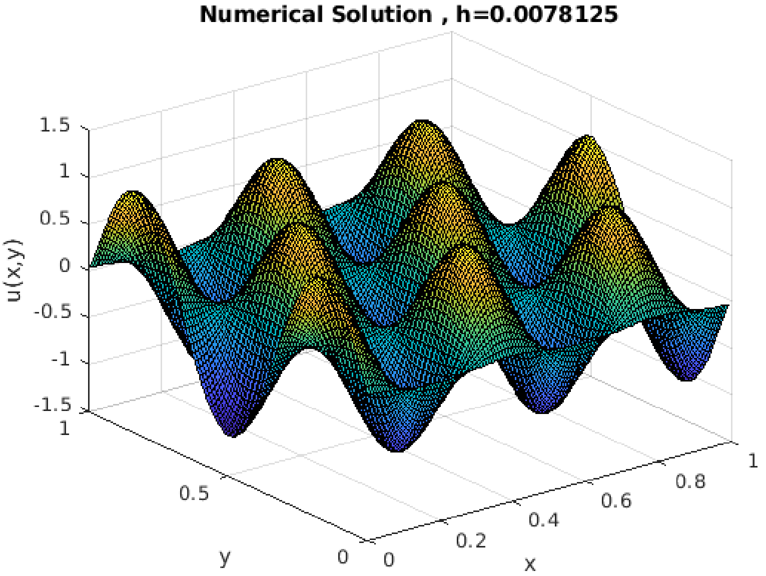
$\sigma(w) = 2w^2 + 2w$

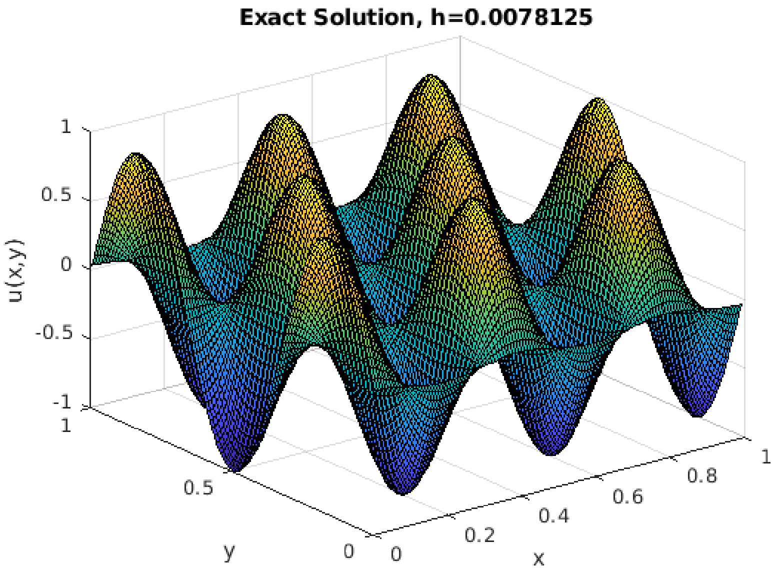
State the number of steps r.

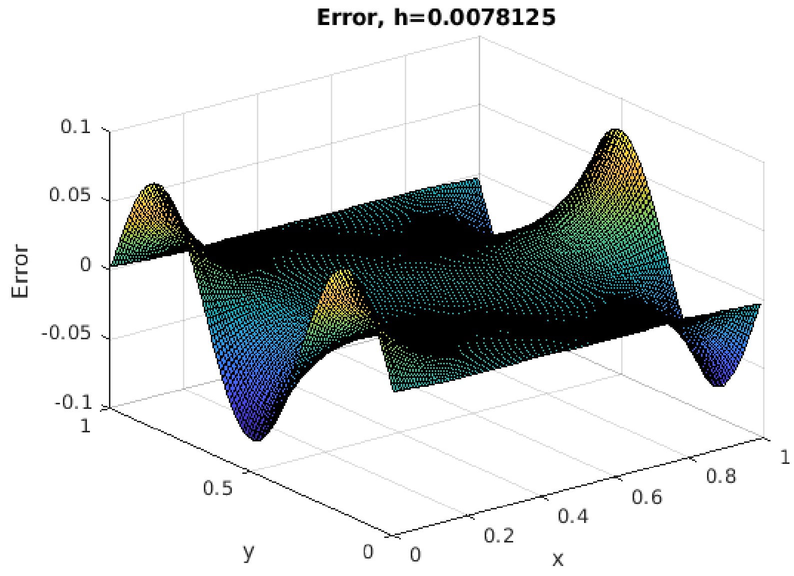
Case	Number of Steps, r,
a)	$r = 2$
b)	$r = 1$
c)	$r = 4$
d)	$r = 3$
e)	$r = 4$
f)	$r = 3$

```
%This program  solves a second order accurate approximaion obtained from
%the poison equation using m = 128

a2 =0; b2 = 1;

m = 128;

f1 = @(x,y) -34*(pi^2)*cos(5*pi*x).*sin(3*pi*y);

%exact solution
uexact = @(x,y) cos(5*pi*x).*(sin(3*pi*y));

h = (b2-a2)/(m); %mesh spacing

w = 2/(1+sin(pi*h)); %optimal relaxation parameter

tol = 10^(-8);   %relative residual

maxiter = 10000;  %maximum value of k

[x,y] = meshgrid(((1:m)-1/2)*h); %Uniform mesh, including boundary points.

dx = 1:m;
dy = 1:m;

u = zeros(m,m);

% Evaluate the RHS of Poisson's equation at the interior points.
f = feval(f1,x(dy,dx),y(dy,dx));


for k = 0:maxiter
    u(:,1) = u(:,2) - (h^2)*f(:,1);
    u(:,m) = u(:,m-1) - (h^2)*f(:,m);
    u(1,:) = (1/3)*(u(2,:)-(h^2)*f(1,:));
    u(m,:) = (1/3)*(u(m-1,:) - (h^2)*f(m,:));
    %Iterate
    for j = 2:(m-1)
        for i = 2:(m-1)
            u(i,j) = (1-w)*u(i,j)+(w/4)*(u(i-1,j)+u(i+1,j)+u(i,j-1)+u(i,j+1)-(h^2)*f(i,j));
        end
    end

    %Compute the residual
    residual = zeros(m,m);

    for j = 2:(m-1)
        for i = 2:(m-1)
            residual(i,j) = -4*u(i,j)+(u(i-1,j)+u(i+1,j)+u(i,j-1)+u(i,j+1)-(h^2)*f(i,j));
        end
    end

     %Determine if convergence has been reached
        if norm(residual(:),2)<tol*norm(f(:),2)
                break
    end
end

error = uexact(x,y) - u;
L2 = R2Norm(error,uexact(x,y));
%polyfit
p=polyfit(log(h),log(L2),1);
p
%fprintf('Since the order of convergence,p, is 2.0014, which is approximately 2, \n hence the method is second order accurate.\n')


% Plot solution
figure, set(gcf,'DefaultAxesFontSize',10,'PaperPosition', [0 0 3.5 3.5]),
surf(x,y,u), xlabel('x'), ylabel('y'), zlabel('u(x,y)'),
title(strcat('Numerical Solution , h=',num2str(h)));

% Plot solution
figure, set(gcf,'DefaultAxesFontSize',10,'PaperPosition', [0 0 3.5 3.5]),
surf(x,y,uexact(x,y)), xlabel('x'), ylabel('y'), zlabel('u(x,y)'),
title(strcat('Exact Solution, h=',num2str(h)));

 %Plot error
figure, set(gcf,'DefaultAxesFontSize',10,'PaperPosition', [0 0 3.5 3.5]),
surf(x,y,u-uexact(x,y)),xlabel('x'),ylabel('y'), zlabel('Error'),
title(strcat('Error, h=',num2str(h)));

function L2 = R2Norm(error, uexact)
R = error .^2;
u_ex = uexact.^2;
L2 = sqrt(sum(R,'all')/sum(u_ex,'all'));
end
```

```
Warning: Polynomial is not unique; degree >= number of data points.

p =

    0.6812         0
```






Error, h=0.0078125
Error
0.1
0.05
0
-0.05
-0.1
1
0.5
0
y
0
0.2
0.4
0.6
0.8
1
x

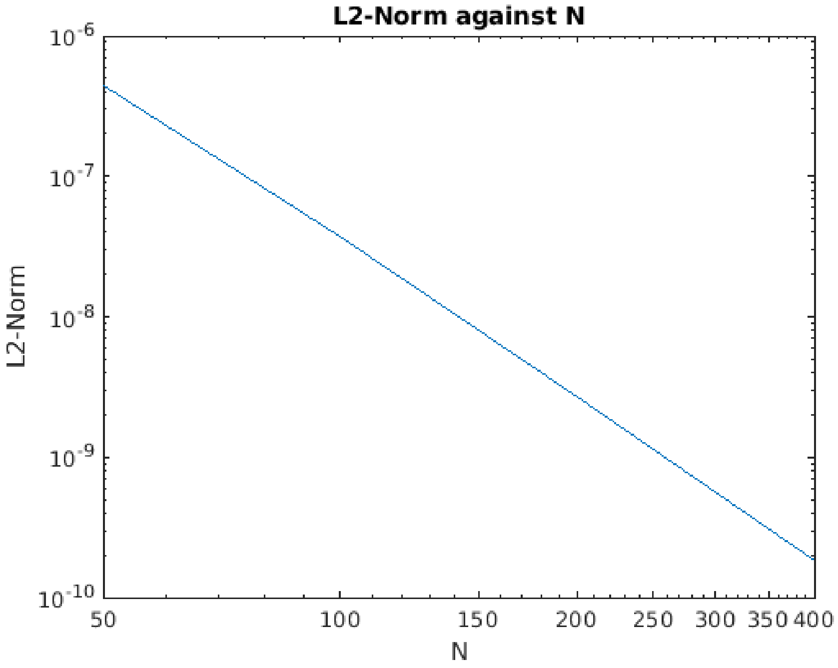
```
% function abm4 solves numerically the general vector-valued IVP.
% It takes in;
% f(t,u) : a vector valued function
% a , b : end points
% uo : initial condition
% N : number of steps
% abm4 outputs;
% t : vector containing all timesteps
% y : matrix containing the numerical solution of all the components of the
% system u at each timestep.

function [t,u] = abm4(f,a,b,uo,N)

% using RK4 to obtain u1, u2, and u3
k = (b-a)/N;

t = zeros(N+1,1);
u = zeros(N+1,1);

t(1) = a;
u(1) = uo;

for j = 1:3
    u1 = u(j);
    u2 = u(j) + (1/2)*k*f(u1,t(j));
    u3 = u(j) + (1/2)*k*f(u2, (t(j) +k/2));
    u4 = u(j) + k*f(u3, (t(j) +k/2));

    t(j+1) = a + j*k;

    u(j+1) = u(j) + (k/6)*(f(u1,t(j)) + 2*f(u2, (t(j) + k/2))...
        + 2*f(u3, (t(j) +k/2)) + f(u4, (t(j) + k)));
end

for n = 4:N
    uast = u(n) + (k/24)*(55*f(t(n),u(n)) - 59*f(t(n-1),u(n-1))...
        + 37*f(t(n-2),u(n-2)) - 9*f(t(n-3),u(n-3)));

    t(n+1) = a + n*k;

    u(n+1) = u(n) + (k/24)*(9*f(t(n-1),uast) + 19*f(t(n),u(n))...
        - 5*f(t(n-1),u(n-1)) + f(t(n-2),u(n-2)));
end

end
```

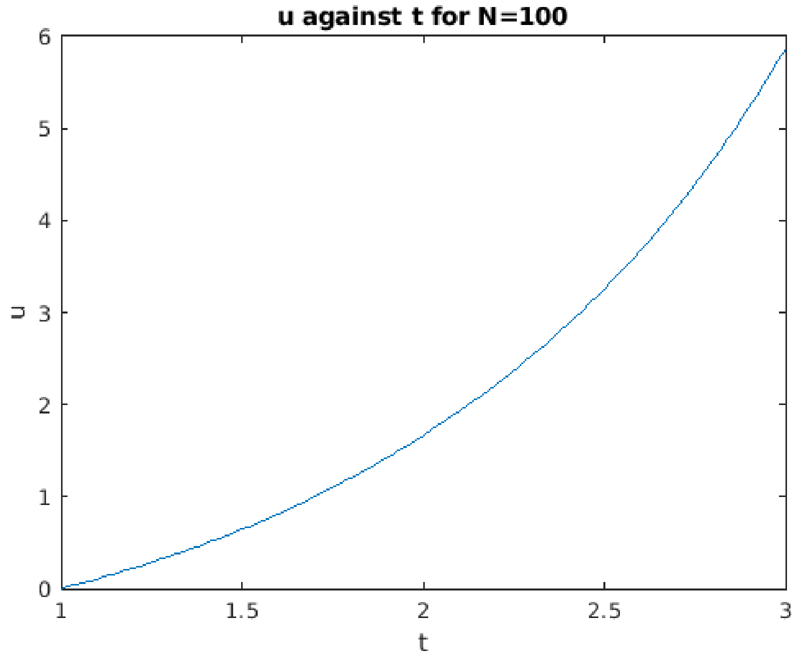
```
Not enough input arguments.

Error in abm4 (line 15)
k = (b-a)/N;
```

Published with MATLAB® R2020a

```
%This program uses abm4 to solve the initial value problem

a=1;b=3;
uo = 0;

N = [50; 100; 200; 400];

f = @(t,u) 1 + (u/t) + (u/t)^2;

%exact solution
uexact = @(t) t.*tan(log(t));

L2 = zeros(4,1);
error = zeros(4,1);

for j = 1:4

    [t,u] = abm4(f,a,b,uo,N(j));

    err = u - uexact(t);

    error(j) = err(N(j) +1);

    L2(j) = L2Norm(uexact(t),u);

end

%Table showing timing results of each method and for each value of m.
t1 = [3;3;3;3];
Table4 = table(t1,N,error, 'VariableNames',{'t','N','Error'})

fprintf('Since as N increases the decreases, hence the larger th N the better convergence of the \n the solution\n');

p = polyfit(log(N),log(L2(:)),1);

fprintf('\nThe order of convergence is %.4f\n',p(1))
fprintf('which is approximately -4, and it is  the same as the slope of the lolog plot\n');

figure(1);
loglog(N,L2);
xlabel('N');
ylabel('L2-Norm');
title('L2-Norm against N');

figure(2);
[t,u] = abm4(f,a,b,uo,N(2));
plot(t,u);
xlabel('t');
ylabel('u');
title('u against t for N=100');

%relative two norm of the error
function L2 = L2Norm(uex,uap)
R = (uex - uap).^2;
L2 = sqrt(sum(R)/sum(uap.^2));
end
```

```
Table4 =

  4×3 table

    t     N       Error
    _    ___    __________

    3     50    3.4945e-06
    3    100    2.9569e-07
    3    200     2.147e-08
    3    400    1.4463e-09

Since as N increases the decreases, hence the larger th N the better convergence of the
 the solution

The order of convergence is -3.7458
which is approximately -4, and it is  the same as the slope of the lolog plot
```





Published with MATLAB® R2020a