Brian KYANJO Homework #5

1. Lagrange polynomials and the Barycentric formula

a) Assume $(x_k, y_k)$, $k = 0, 1, \dots, N$

Write $P_2(x)$ for $N = 2$ using the Lagrange Interpolation formula.

Given

$$L_j(x) = \frac{\prod_{k=0, k\neq j}^{n} (x - x_k)}{\prod_{k=0, k\neq j}^{n} (x_j - x_k)} \quad \text{--- (1)}$$

$$P_2(x) = \sum_{j=0}^{N} L_j(x) y_j = L_0(x) y_0 + L_1(x) y_1 + L_2(x) y_2 \quad \text{--- (2)}$$

from Equation (1)

$$L_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}$$

$$L_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}$$

$$L_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

therefore Equation (2) becomes

$$P_2(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} y_0 + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} y_1 + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} y_2$$

b) How many operations are required to evaluate $P_2(x)$

Since the highest term in $P_2(x)$ will be of power 2, then the number of operations for a second order polynomial (quadratic) will be given by $N^2$, hence $2^2 = \underline{4 \text{ operations}}$.

c) The second Barycentric form is given by

$$P_N(x) = \frac{\sum_{j=0}^{N} \frac{w_j}{x - x_j} y_j}{\sum_{j=0}^{N} \frac{w_j}{x - x_j}}$$

where weights are computed as

$$w_j = \frac{1}{\prod_{\substack{k=0 \\ k \neq j}}^{N} (x_j - x_k)}, \quad j = 0, 1, \ldots, N.$$

For $N=2$,

$$P_2(x) = \frac{\sum_{j=0}^{N=2} \frac{w_j}{x - x_j} y_j}{\sum_{j=0}^{N=2} \frac{w_j}{x - x_j}}, \quad w_j = \frac{1}{\prod_{\substack{k=0 \\ k \neq j}}^{2} (x_j - x_k)}$$

$$P_2(x) = \frac{\frac{w_0}{x - x_0} y_0 + \frac{w_1}{x - x_1} y_1 + \frac{w_2}{x - x_2} y_2}{\frac{w_0}{x - x_0} + \frac{w_1}{x - x_1} + \frac{w_2}{x - x_2}}$$

$$P_2(x) = \frac{w_0 y_0 (x-x_1)(x-x_2) + w_1 y_1 (x-x_0)(x-x_2) + w_2 y_2 (x-x_0)(x-x_1)}{w_0 (x-x_1)(x-x_2) + w_1 (x-x_0)(x-x_2) + w_2 (x-x_0)(x-x_1)}$$

Since

$$w_j = \frac{1}{\prod_{\substack{k=0 \\ k \neq j}}^{2} (x_j - x_k)}, \quad w_0 = \frac{1}{(x_0-x_1)(x_0-x_2)}$$

$$w_1 = \frac{1}{(x_1-x_0)(x_1-x_2)}$$

$$w_2 = \frac{1}{(x_2-x_0)(x_2-x_1)}$$

therefore $P_2(x)$ becomes

$$P_2(x) = \frac{\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} y_0 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} y_1 + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} y_2}{\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}}$$

substituting for $L_j$, $j = 0, 1, 2$, we get

$$P_2(x) = \frac{L_0 y_0 + L_1 y_1 + L_2 y_2}{L_0 + L_1 + L_2}$$

but $\sum_{j=0}^{2} L_j = 1$

$$P_2(x) = L_0 y_0 + L_1 y_1 + L_2 y_2$$

which is the exact form of equation (2),

d) Only 2 operations are required.

e) For general $N$, The Lagrange polynomial is given by

$$P_N(x) = \sum_{j=1}^{N} f_j \cdot \prod_{\substack{k=1 \\ k \neq j}}^{N} \frac{x - x_k}{x_j - x_k} \quad — ①$$

Equation can be converted into.

$$P_N(x) = \prod_{k=1}^{N} (x - x_k) \cdot \sum_{j=1}^{N} \frac{f_j}{x - x_j} \prod_{\substack{k=1 \\ k \neq j}}^{N} \frac{1}{x_j - x_k} \quad — ②$$

at $N$ points Equation ② can be evaluate as

$$O\left(N \cdot \log\left(\frac{1}{\epsilon}\right)\right),$$

This requires $O(N^2)$ operations to be evaluated, which is not the same case for the Barycentric form

- The ~~IBA~~ $$P_N(x) = \frac{\sum_{j=0}^{N} y_j \prod_{\substack{k=0 \\ k\neq j}}^{N} \frac{(x_j - x_k)}{x - x_j}}{\sum_{j=0}^{N} \prod_{\substack{k=0 \\ k \neq j}}^{N} \left(\frac{x_j - x_k}{x - x_j}\right)}, \; j = 0, 1, \ldots$$

$P_N(x)$ Can also be evaluate as

$$\cancel{O\left(N \cdot \log\left(\frac{1}{\epsilon}\right)\right)} \approx O(N^2) \text{ operations}$$

The Barycentric Form is more efficient.

$$P_N(x) = \frac{\sum_{j=0}^{N} y_j \cdot \prod_{\substack{k=0 \\ k \neq j}}^{N} \left( \frac{x_j - x_k}{x - x_j} \right)}{\sum_{j=0}^{N} \prod_{\substack{k=0 \\ k \neq j}}^{N} \left( \frac{x_j - x_k}{x - x_j} \right)}$$

this can be evaluated as $\frac{O(N \cdot \log(\frac{1}{\epsilon}))}{O(N)}$

which becomes $\frac{O(N^2)}{O(N)} \approx O(N)$

Hence Barycentric form requires N op O(N) operat

Therefore: The Barycentric form is more efficient

NO. 2 (b)

(ii) Why does the newton iteration Converge in one step?

Given
$$d^{k+1} = d^k - J^{-1} F(d^k)$$

Since the Jacobian $J$ is given by the Hessian $\nabla f(d)$ i.e. $J = HF(d)$,

For the newton iteration $F$ is strictly convex and has a unique strict global minimizer $d^*$,

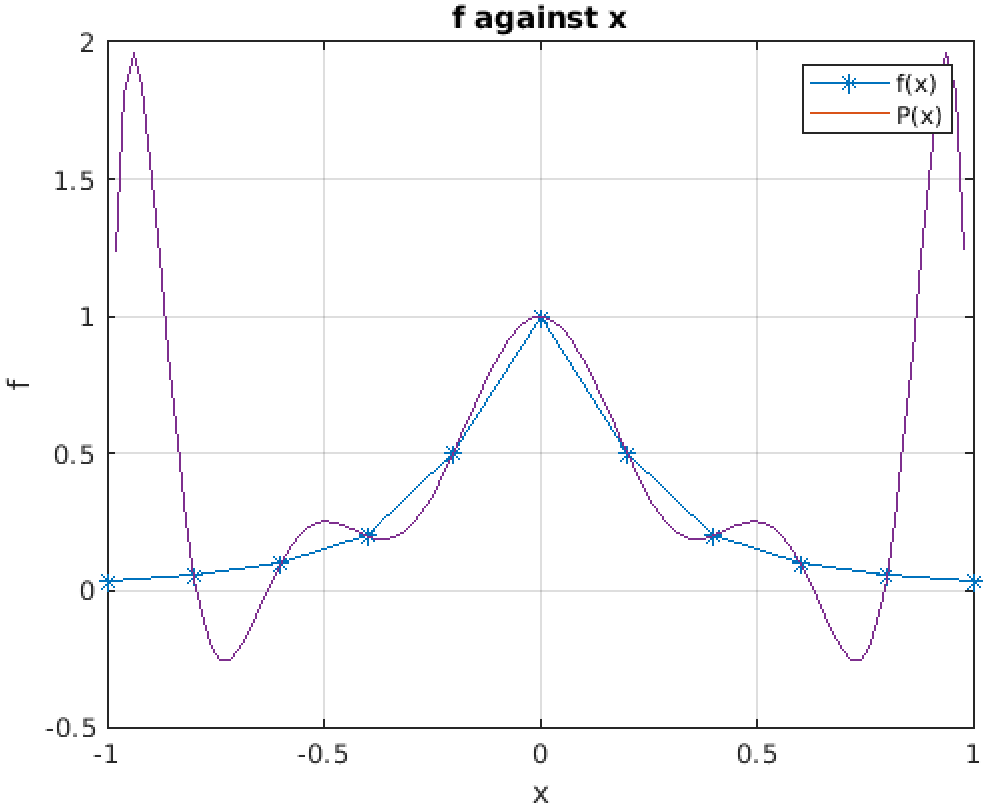
$$A d^* = -b$$

where $A$ is an $n \times n$ positive symmetric matrix

$F(d) = a + b \cdot d + \frac{1}{2} d \cdot A d$,

~~F(d*)~~ The newton iteration approximates $F(d^*)$ quadratically near $d^k$, therefore for any initial guess $d^{(0)}$, the Newton's ~~…~~ iteration applied to $F(d)$ Converges to $d^*$ in one step.

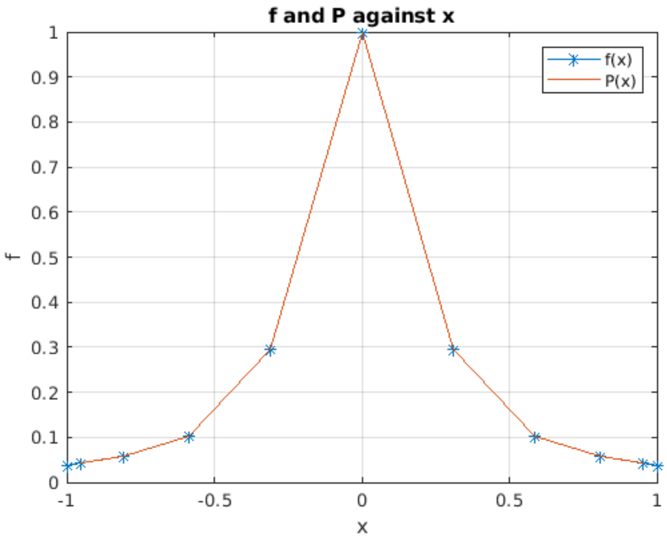
```
% The code approximates the function f(x) with a 10th polynomial.

clear all
close all

N = 10; h = 2/N; m = 100;

xx = linspace(-1,1,m)';

%function
f1 = @(x) 1./(1+25*x.^2);

x = zeros(1,N+1);
f = zeros(1,N+1);
for k = 0:N
   x(k+1) = -1 + k*h;
   f(k+1) = f1(x(k+1));
end

PN = [];
for k = 1:m+1
     pN = Barycentric(x,f,xx,N);
     PN = [PN,pN];
end

plot(x,f1(x),'-*'); grid on;
title('f against x');
xlabel('x');ylabel('f');
hold on
plot(xx,PN);
legend('f(x)','P(x)');


function pN = Barycentric(x,f,xx,N)

%weights
w = zeros(N+1,1);

numer = 0; %Numerator
demon = 0; %Denominator
for j = 1:N+1
    temp = 1;
    for k = 1:N+1
        if (k ~= j)
            temp = temp*(x(j) - x(k));
        end
    end
    w(j) = 1/temp;

    xdiff = xx-x(j);
    temp1 = w(j)./(xdiff);

    numer = numer + temp1*f(j);
    demon = demon + temp1;
end
pN = numer./demon;
end
```




*Published with MATLAB® R2020a*

```
% The code approximates the function f(x) with a 10th polynomial.

clear all
close all

N = 10; h = 2/N; m = 300;

%function
f1 = @(x) 1./(1+25*x.^2);

x = zeros(1,N+1);
f = zeros(1,N+1);
for k = 0:N
   x(k+1) = -cos(k*pi/10);
   f(k+1) = f1(x(k+1));
end

xk = [];
PN = [];
for k = 1:m+1
     xk1 = -cos(k*pi/10); xk = [xk,xk1];
     pN = Barycentric(x,f,xk(k),N);
     PN = [PN,pN];
end

plot(x,f1(x),'-*'); grid on;
title('f and P against x');
xlabel('x');ylabel('f');
hold on
plot(xk,PN);
legend('f(x)','P(x)');

%compare
fprintf('Using the Chebyshev nodes its clear that the intepolant curve fits the data well, hence\n Chebyshev nodes approximate better than the ec

function pN = Barycentric(x,f,xx,N)

%weights
w = zeros(N+1,1);

numer = 0; %Numerator
demon = 0; %Denominator
for j = 1:N+1
    temp = 1;
    for k = 1:N+1
        if (k ~= j)
            temp = temp*(x(j) - x(k));
        end
    end
    w(j) = 1/temp;

    xdiff = xx-x(j);
    temp1 = w(j)./(xdiff);

    numer = numer + temp1*f(j);
    demon = demon + temp1;
end
pN = numer./demon;
end
```

```
Using the Chebyshev nodes its clear that the intepolant curve fits the data well, hence
 Chebyshev nodes approximate better than the equispaced points
```



```
function pp  = math565_build_spline(xd,yd,ec,ed)

global xdata ydata end_cond end_data

% Set up global variables needed for other routines
xdata = xd;
ydata = yd;
end_cond = ec;
end_data = ed;

% Solve for the end point derivatives d = [d0,d1] using Newton's Method
d0 = [0;0];  % Starting value

% TODO : Compute the Jacobian J = F'(d), 2x2 matrix

J = zeros(2,2);

% Get column 1 of J :
 J(:,1) =  F(d0 + [1;0]) - F(d0);

% Get column two of J :
J(:,2) = F(d0 + [0; 1]) - F(d0);

% TODO : Setup Newton iteration to solve for d = [d0,d1]
dk = [0;0];  % Starting value

% Newton iteration :

dk = dk -J\F(dk);

d = dk;    % Change this to the correct value for d

% Compute derivatives using d=[d(1),d(2)] from above
ddata = compute_derivs(d);

% Get the spline coefficients
coeffs = spline_coeffs(ddata);

% Use Matlab function mkpp to set up spline
pp = mkpp(xdata,coeffs);

end

% Function F(d) :    Use Newton's method to solve F(d) = 0
function Fd = F(d)

global xdata ydata end_cond end_data

ddata = compute_derivs(d);

% Spline coefficients for this choice of end point values
coeffs = spline_coeffs(ddata);

% Four coefficients for the cubic in the first segment :
% p0(x) = a(1)*(x-x0)^3 + a(2)*(x-x0)^2 + a(3)(x-x0) + a(4)
a = coeffs(1,:);

% Four coefficients for the cubic in the last segment :
% p_{N-1}(x) = b(1)*(x-x_N-1)^3 + b(2)*(x-x_N-1)^2 + b(3)(x-x_N-1) + b(4)
b = coeffs(end,:);

% TODO : Use a and b to compute the first and second derivatives
% of the spline interplant at the endpoints  x_0 and x_N
```

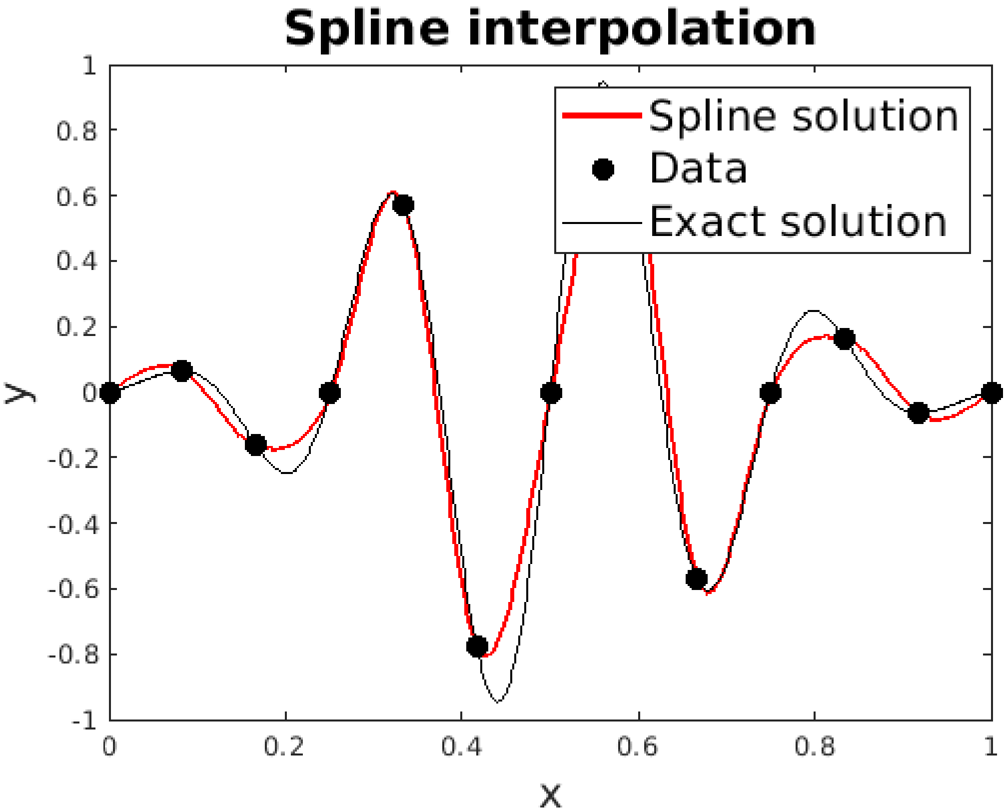
```
p0_deriv_x0 = a(3);
p0_deriv2_x0 = 2*a(2);

 pNm1_deriv_xN = 3*b(1)*(xdata(end) - xdata(end-1))^2 + 2*b(2)*(xdata(end)-xdata(end-1)) + b(3);
 pNm1_deriv2_xN =  6*b(1)*(xdata(end) - xdata(end-1)) + 2*b(2);

% TODO : Define function F(d) :
Fd = [0;0];
switch end_cond
    case 'natural'
       Fd(1) = p0_deriv2_x0;
       Fd(2) = pNm1_deriv2_xN;
    case 'clamped'
        Fd(1) = d(1);
        Fd(2) = d(2);
end

end

% Given values of d = [d0,d1], compute derivatives
% at all nodes. This will require a linear solve.
function ddata = compute_derivs(d)

global xdata ydata

N = length(xdata) - 1;
h = diff(xdata);

% Compute the derivatives at the internal nodes

% TODO : Set up a linear system to solve for interval derivatives
 A = zeros(N-1);

 for j=1:N-1
    A(j,j)= 2*(1/h(j)+1/h(j+1));
end

for j=2:N-1
    A(j,j-1) = 1/j;
    A(j-1,j)=1/j;
end


% TODO : Set up right hand side vector b
 b = zeros(N-1,1);

 for i = 2:N
    b(i-1) = 3*(1/h(i-1))^2*(ydata(i)- ydata(i-1)) + 3*(1/h(i))^2*(ydata(i+1) - ydata(i));
 end

% TODO : Solve for dH.  Use either 'backslash', or one of the other
% routines you learned in class.

dH = A\b;

% Construct vectors needed for endpoint conditions
u0 = [h(1); zeros(N-2,1)];
uN = [zeros(N-2,1); h(end)];

% Augment internal nodes with endpoint derivatives
ddata = [d(1); dH - d(1)*u0 - d(2)*uN; d(2)];
end


% Spline coefficients for each segment.  See page 14 of Lecture
% notes on Piecewise Polynomials
```

```
function coeffs = spline_coeffs(ddata)

global xdata ydata

N = length(xdata) - 1;

h = diff(xdata);
C = [-2, 3, 0, -1; 2, -3, 0, 0; -1, 2, -1, 0; -1, 1, 0, 0];
coeffs = zeros(N,4);
for k = 1:N
    p = [ydata(k); ydata(k+1); ddata(k)*h(k); ddata(k+1)*h(k)];
    S = diag(1./h(k).^(3:-1:0));
    coeffs(k,:) = -p'*C*S;
end

end
```

```
Not enough input arguments.

Error in math565_build_spline (line 6)
xdata = xd;
```

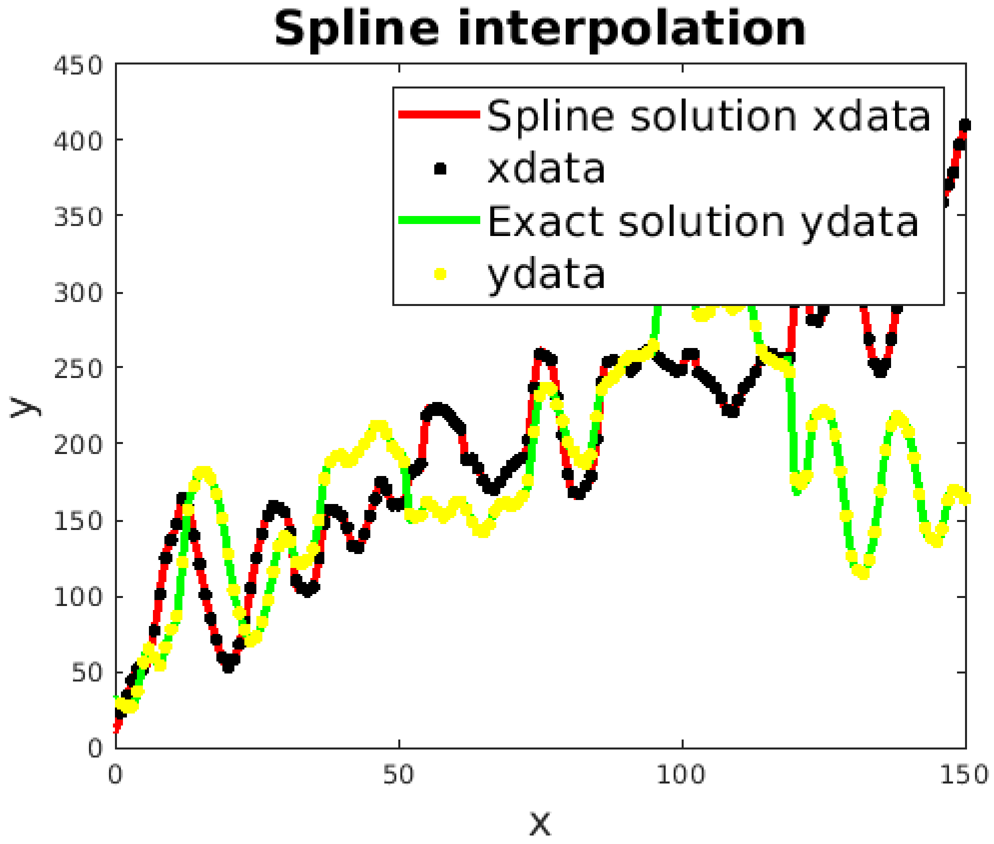
*Published with MATLAB® R2020a*

```
function spline_test()

% Function to interpolate
a = 15;
m = 8;
b = 1/2;
f = @(x) exp(-a*(x-b).^2).*sin(m*pi*x);
% Derivative (ne
fp = @(x) exp(-a*(x-b).^2).*(m*pi*cos(m*pi*x) - 2*a*(x-b)*sin(m*pi*x));

% Data at equispaced points
N = 12;
xdata = linspace(0,1,N+1);
ydata = f(xdata);

% Build the spline.  Currently, the derivatives at nodes are all set to
% zero.  Your job is to come up with a nicer spline by modifying
% the routines below.
math465 = false;
if (math465)
    pp = math465_build_spline(xdata,ydata);
else
    end_cond = 'natural';                          % 'natural' or 'clamped'
    end_data = [fp(xdata(1)); fp(xdata(end))]';   % For 'clamped' endpoint condition

    pp = math565_build_spline(xdata,ydata,end_cond,end_data);
end

% Evaluate the spline at points used for plotting
xv = linspace(0,1,500);
yv = ppval(pp,xv);       % Matlab function

% Plot results
figure(2)
clf;

% Plotting
plot(xv,yv,'r','linewidth',2);
hold on;
plot(xdata,ydata,'k.','markersize',30);
plot(xv,f(xv),'k');
legend('Spline solution','Data','Exact solution','fontsize',16);

xlabel('x','fontsize',16);
ylabel('y','fontsize',16);
title('Spline interpolation','fontsize',18);

shg

end
```




*Published with MATLAB® R2020a*

```
function spline_test()

%load XY_dots.dat
[filename directory_name] = uigetfile('*.dat', 'Select a file');
XY = load(fullfile(directory_name, filename));

% Data at equispaced points
k = XY(:,1);
xdata = XY(:,2);
ydata = XY(:,3);

% Build the spline.  Currently, the derivatives at nodes are all set to
% zero.  Your job is to come up with a nicer spline by modifying
% the routines below.
math465 = false;
if (math465)
    pp = math465_build_spline(xdata,ydata);
else
    end_cond = 'natural';                         % 'natural' or 'clamped'
    end_data = [xdata(1); ydata(end)]';   % For 'clamped' endpoint condition
    ppx = math565_build_spline(k,xdata,end_cond,end_data);
    ppy = math565_build_spline(k,ydata,end_cond,end_data);
end

% Evaluate the spline at points used for plotting
v = linspace(0,150,2000);
x_k = ppval(ppx,v); y_k = ppval(ppy,v);    % Matlab function

% Plot results
figure(2)
clf;

% Plotting
plot(v,x_k,'r','linewidth',3);
hold on;
plot(k,xdata,'k.','markersize',12);
plot(v,y_k,'g','linewidth',3);
plot(k,ydata,'y.','markersize',12);
legend('Spline solution xdata','xdata','Exact solution ydata','ydata','fontsize',16);

xlabel('x','fontsize',16);
ylabel('y','fontsize',16);
title('Spline interpolation','fontsize',18);

shg

end
```



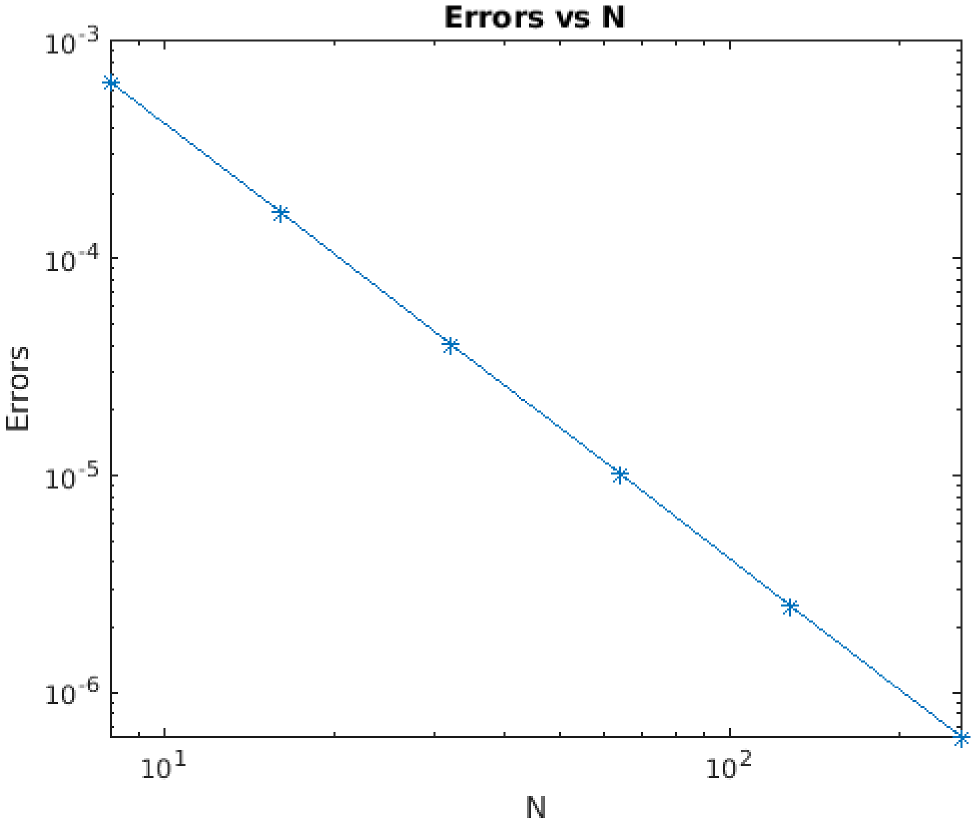

---

*Published with MATLAB® R2020a*

```
% The program uses trapezoidal rule to evaluate the arc length along an
% ellipse. at t=b=1

clear all
close all

a = 0; b = 1;
A = 1; B = 0.5;
k = sqrt(1 - (B/A)^2);
f = @(x) A*(sqrt(1 - k^2*(sin(x)).^2));

%exact solution
Tex =  0.8866251235367069482;

n = [8,16,32,64,128,256];
c = length(n);
Error = [];

for i = 1:c

    T= trapezoidal(a,b,f,n(i));
    error = abs(T-Tex);
    Error = [Error,error];

end

%Table of errors
Table = table(n(:),Error(:),'VariableNames',{'N','Error'})

%loglog plot
loglog(n,Error,'-*');
title('Errors vs N');
xlabel('N'); ylabel('Errors');

%order of convergence
p = polyfit(log(n),log(Error),1); p(1)

fprintf('Hence order of convergence is 2\n');

function [T] = trapezoidal(a,b,f,n)
    h = (b-a)/n;
    xe = linspace(a,b,n+1); %Nodes at edges

    fe = f(xe);

    T = (h/2)*(fe(1) + 2*sum(fe(2:end-1)) + fe(end));
end
```

```
Table =

  6×2 table

     N       Error
    ___    __________

      8     0.0006491
     16    0.00016214
     32    4.0525e-05
     64    1.0131e-05
    128    2.5327e-06
    256    6.3316e-07
```

```
ans =

   -2.0003

Hence order of convergence is 2
```

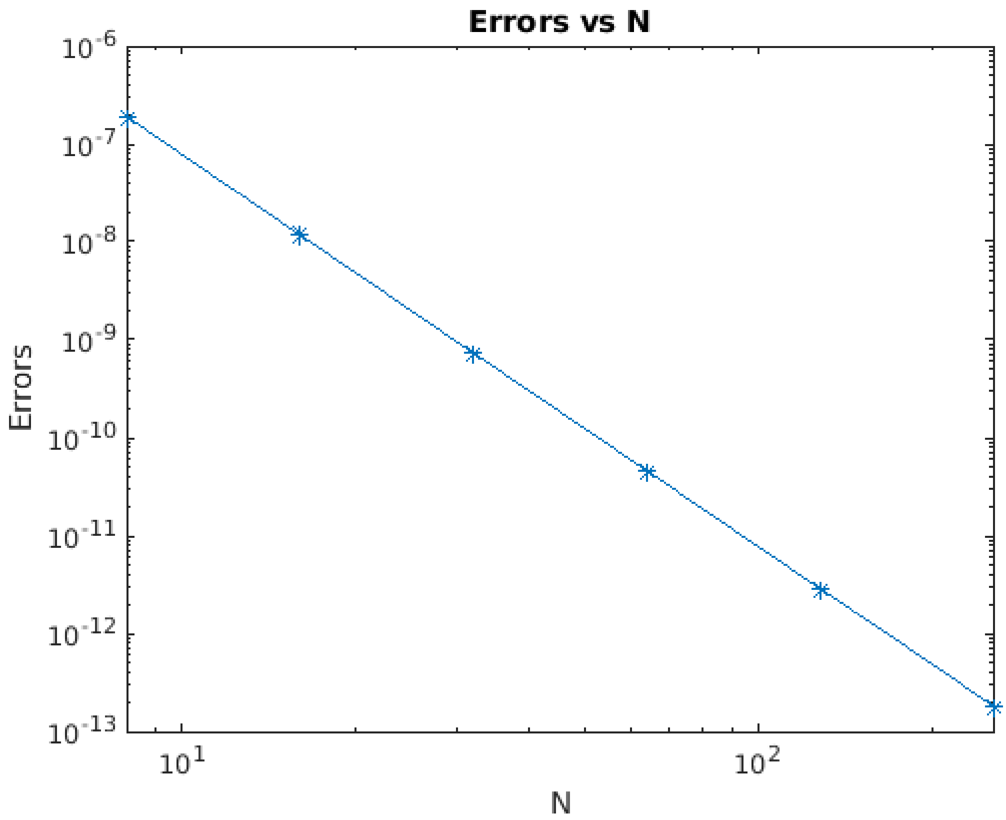



*Published with MATLAB® R2020a*

```
% The program uses Simpson's rule to evaluate the arc length along an
% ellipse. at t=b=1

clear all
close all

a = 0; b = 1;
A = 1; B = 0.5;
k = sqrt(1 - (B/A)^2);
f = @(x) A*(sqrt(1 - k^2*(sin(x)).^2));

%exact solution
Tex =  0.8866251235367069482;

n = [8,16,32,64,128,256];
c = length(n);
Error = [];

for i = 1:c

    S= simpson(a,b,f,n(i));
    error = abs(S-Tex);
    Error = [Error,error];

end

%Table of errors
Table = table(n(:),Error(:),'VariableNames',{'N','Error'})

%loglog plot
loglog(n,Error,'-*'); xlim('auto');
title('Errors vs N');
xlabel('N'); ylabel('Errors');

%order of convergence
p = polyfit(log(n),log(Error),1); p(1)

fprintf('Hence order of convergence is 4\n');

function [S] = simpson(a,b,f,n)
    h = (b-a)/n;
    xe = linspace(a,b,n+1); %Nodes at edges
    xc = xe(1:end-1) + h/2; %Nodes at centers

    fe = f(xe);
    fc = f(xc);

    M = h*sum(fc);

    T = (h/2)*(fe(1) + 2*sum(fe(2:end-1)) + fe(end));

    S = (T + 2*M)/3;
end
```

```
Table =

  6×2 table

     N       Error
    ___    __________

      8    1.8594e-07
```

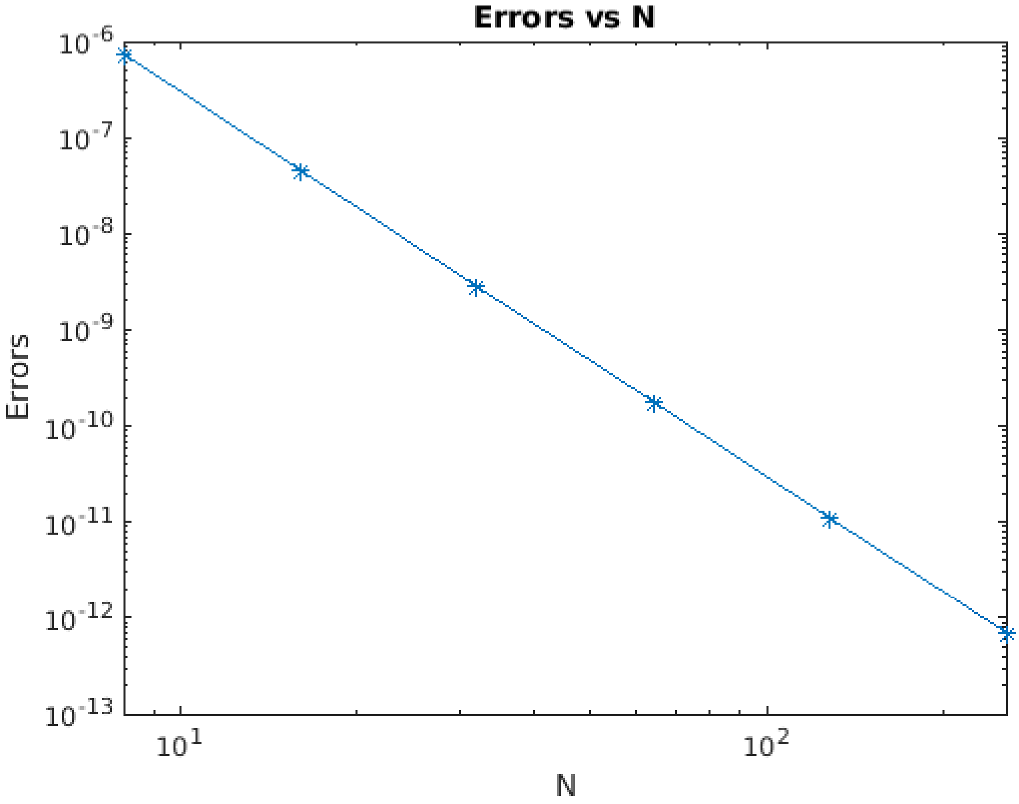
```
      16    1.1632e-08
      32    7.2718e-10
      64    4.5451e-11
     128    2.8406e-12
     256    1.7719e-13


ans =

   -4.0001

Hence order of convergence is 4
```




*Published with MATLAB® R2020a*

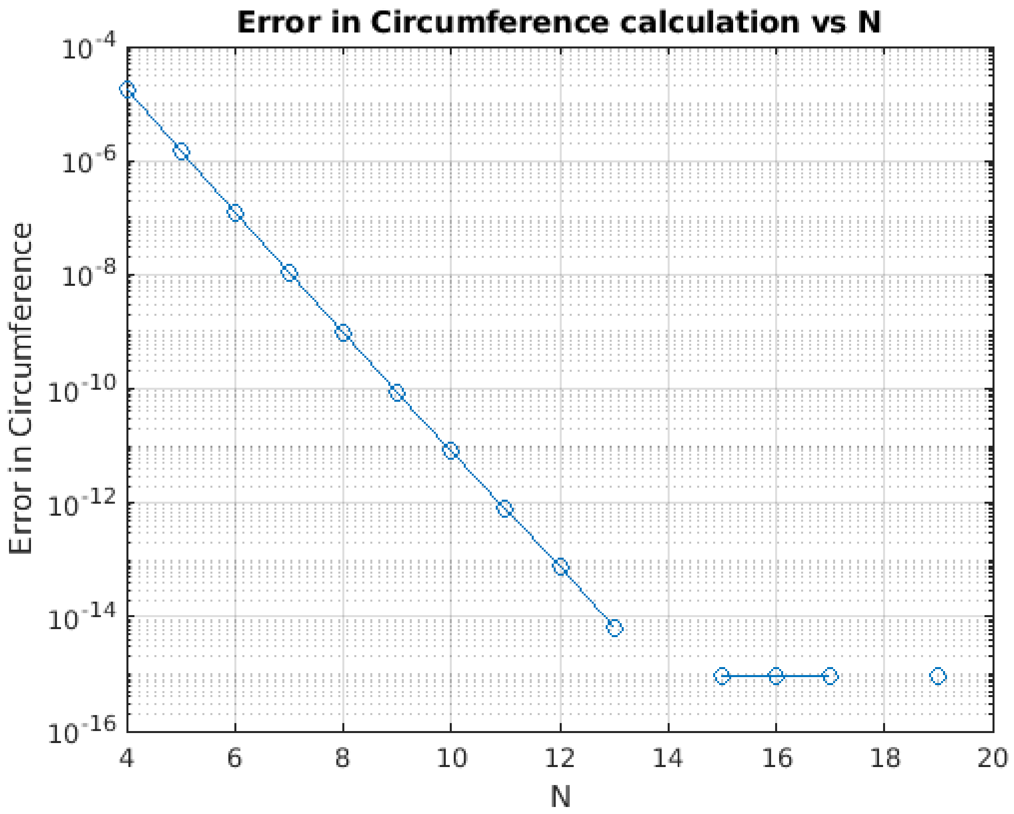
```
% This program corrects the trapezoidal method to obtain fourth order
% method.

clear all
close all

a = 0; b = 1;
A = 1; B = 0.5;
k = sqrt(1 - (B/A)^2);

f = @(x) A*(sqrt(1 - k^2*(sin(x)).^2));
fp = @(x) -A*k^2*sin(2*x).*(2*sqrt(1 - k^2*(sin(x)).^2)).^-1; %fprime

%exact solution
Tex =  0.8866251235367069482;

n = [8,16,32,64,128,256];
c = length(n);
Error = [];

for i = 1:c

    Tc= trape(a,b,f,fp,n(i));
    error = abs(Tc-Tex);
    Error = [Error,error];

end

%loglog plot
loglog(n,Error,'-*');
title('Errors vs N');
xlabel('N'); ylabel('Errors');

%order of convergence
p = polyfit(log(n),log(Error),1); p(1)

fprintf('Hence order of convergence is 4\n');


function [Tc] = trape(a,b,f,fp,n)
    h = (b-a)/n;
    xe = linspace(a,b,n+1); %Nodes at edges

    fe = f(xe);

    T = (h/2)*(fe(1) + 2*sum(fe(2:end-1)) + fe(end));

    fpp = fp(xe);

    Tc = T - ((h^2)/12)*(fpp(end) - fpp(1));
end
```

```
ans =

   -3.9998

Hence order of convergence is 4
```

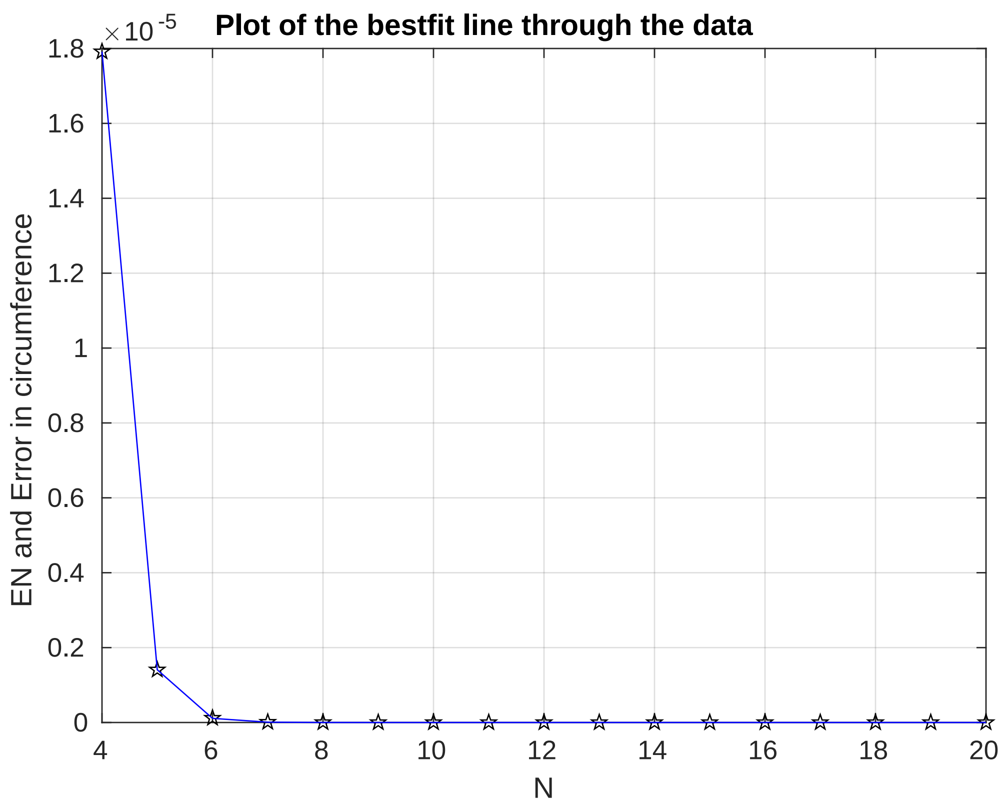



*Published with MATLAB® R2020a*

```
% This program evaulates the circumference of the ellipse using the trapezoidal rule

clear all
close all

a = 0; b = pi/2;
A = 1; B = 0.5;
k = sqrt(1 - (B/A)^2);

f = @(x) 4*A*(sqrt(1 - k^2*(sin(x)).^2));
%fp = @(x) -2*A*k^2*sin(2*x).*(sqrt(1 - k^2*(sin(x)).^2)).^-1; %fprime


% exact
Tex =  4.84422411027383809921;

Error = [];
C = [];
N = [];
for n = 4:20

    N = [N,n];
    Tc = trapezoidal(a,b,f,n);
    C = [C,Tc];
    error = abs(Tc-Tex);
    Error = [Error,error];

end

%log-linear plot
semilogy(N,Error,'-o'); grid on;
title('Error in Circumference calculation vs N');
xlabel('N'); ylabel('Error in Circumference');

%composite trapezoidalrule
function [T] = trapezoidal(a,b,f,n)
    h = (b-a)/n;
    xe = linspace(a,b,n+1); %Nodes at edges

    fe = f(xe);

    T = (h/2)*(fe(1) + 2*sum(fe(2:end-1)) + fe(end));
end
```



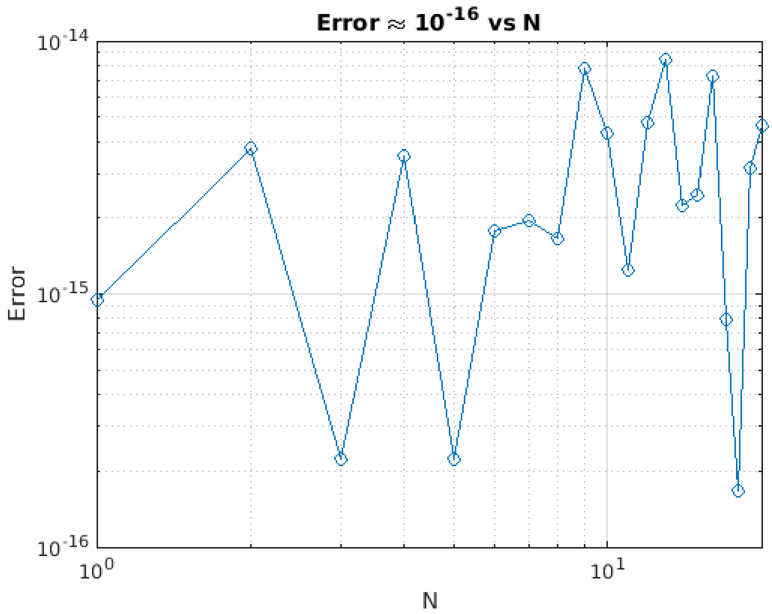

*Published with MATLAB® R2020a*

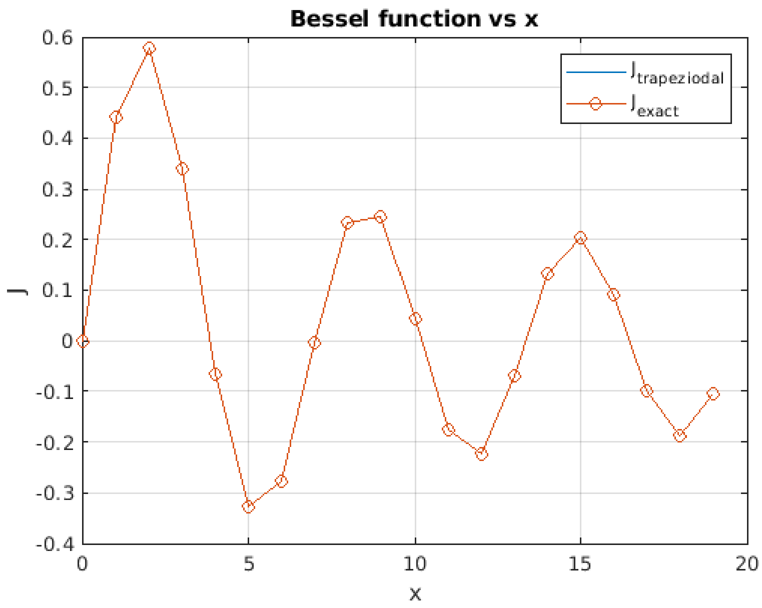


Plot of the bestfit line through the data
×10^-5
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
EN and Error in circumference
4
6
8
10
12
14
16
18
20
N

```
clear all;
close all;

J =@(x,t) 1/pi*cos(t -x*sin(t));

N = 20; h = 20/N;  a=0; b=pi; n = 1e6; h1 = (b-a)/n;
t = linspace(0,pi,n+1);

x = [];
T = [];
for i = 0:19
    x1 = i*h;
    x = [x,x1];
end

for i = 1:20
     %trapezoidal rule
    T1 = (h1/2)*(J(x(i),t(1))+2*sum(J(x(i),t(2:n)))+J(x(i),t(end)));
    T = [T,T1];
end

%exact
J1=besselj(1,x);
error = abs(J1 - T);

%table
N = [1:20]';
Table = table(N(:),error(:),'VariableNames',{'N','Error'})

fprintf('Hence error values are approximately 10^-16');

%convergence
fprintf('Exponential convergence, due to the fact that we are dealing with a periodic integral');

figure(1)
loglog(N,error, '-o'); grid on
xlabel('N');ylabel('Error');
title('Error \approx 10^-^1^6 vs N');

figure(2)
plot(x,T); grid on
xlabel('x');ylabel('J');
title('Bessel function vs x');
hold on
plot(x,J1,'-o');
legend('J_t_r_a_p_e_z_i_o_d_a_l','J_e_x_a_c_t')
```

```
Table =

  20×2 table

    N       Error
    __    __________

     1    9.4459e-16
     2    3.7748e-15
     3    2.2204e-16
     4    3.4972e-15
     5    2.2204e-16
     6    1.7764e-15
     7    1.9429e-15
     8    1.6575e-15
     9    7.7438e-15
    10    4.3299e-15
    11    1.2421e-15
    12    4.7462e-15
    13    8.4099e-15
    14    2.2343e-15
    15    2.4702e-15
    16    7.2442e-15
    17    7.9103e-16
    18    1.6653e-16
    19    3.1641e-15
    20    4.6213e-15

Hence error values are approximately 10^-16Exponential convergence, due to the fact that we are dealing with a periodic integral
```




---

*Published with MATLAB® R2020a*