Guass Quadrature

Table of Contents

Exercise 1

In this exercise I have created a function that numerically approximates the integral of x^k for $x \in [-1, 1]$ using a 3-point quadrature formula that I will denote as I_3. In this first section I will show that $I_3 = I$, where I is the actual integral, for $0 \leq k \leq 5$ but not for $k = 6$.

```
syms x
for i = 0:6 % For loop to go through each k
    integral = int(x^i,x,-1,1); % Actual value of the integral
    approxintegral = threepointformula(i); % Numerical approximation
    count = num2str(i); % Will be used in display
    if integral == approxintegral % Checks to see if the integrals are the
 same
        X = ['Exact for k = ',count]; % Disp only takes one argument
        disp(X) % Shows for what value of k this was
    else
        X = ['Not exact for k = ',count];
        disp(X)
    end
end
```

```
Exact for k = 0
Exact for k = 1
Exact for k = 2
Exact for k = 3
Exact for k = 4
Exact for k = 5
Not exact for k = 6
```

This showed that $I_3 = I$ for $0 \leq k \leq 5$ but not for $k = 6$. In this next section I will show the errors, $E_n = I_n - I$ for $0 \leq k \leq 10$.

```
threepointerror()
```

```
The error when k = 0 is 0
The error when k = 1 is 0
The error when k = 2 is 0
```

```
The error when k = 3 is 0
The error when k = 4 is 0
The error when k = 5 is 0
The error when k = 6 is -0.045714
The error when k = 7 is 0
The error when k = 8 is -0.078222
The error when k = 9 is 0
The error when k = 10 is -0.095418
```

It is clear that for $k = 7, 8, 9, 10$ that I_3 does not always equal I. However, when k is odd, $I_3 = I$. This is because x^k is an even function when k is even and an odd function when k is odd. Therefore, the integral for $x \in [-1, 1]$ when x^k is odd will be 0. By looking at the 3-point quadrature formula we can see that I_3 always equals 0 when k is odd.

Exercise 2

In this next section I have created a function that returns an $nx2$ vector where the first column contains nodes and the second column contains the corresponding weights for an $n - point$ quadrature formula. I will verify that for $n = 3$ that the nodes and corresponding weights are the same from the previous question.

```
[x,w] = guassq(3);
disp(sym([x,w]))
```

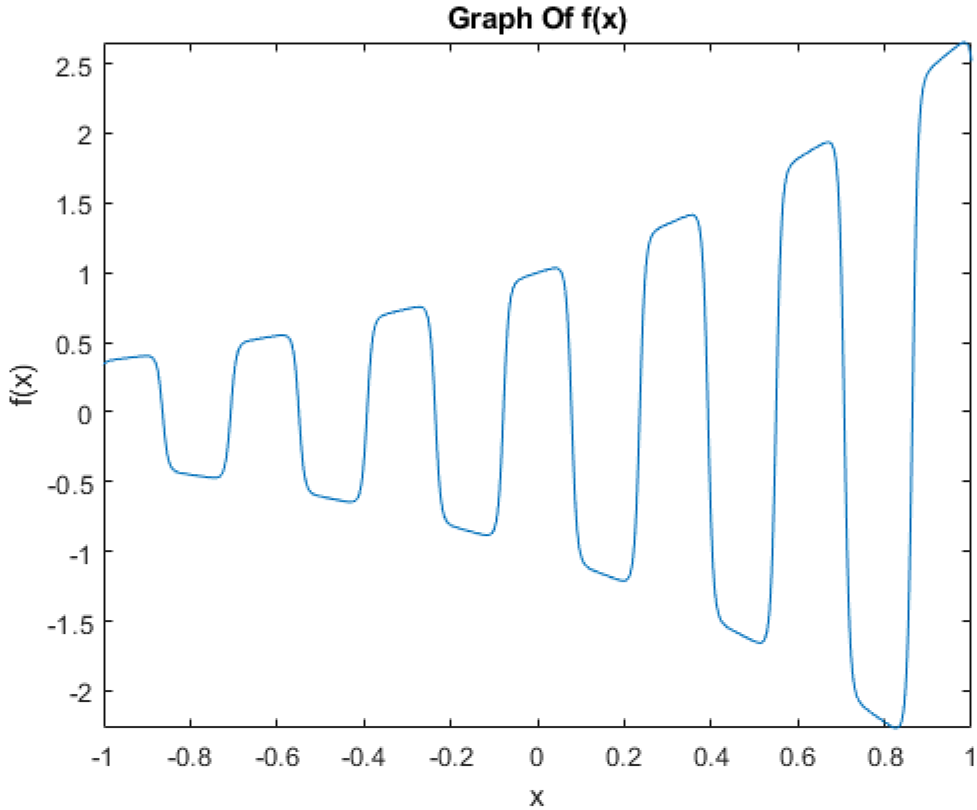
```
[-(3^(1/2)*5^(1/2))/5, 5/9]
[                   0, 8/9]
[ (3^(1/2)*5^(1/2))/5, 5/9]
```

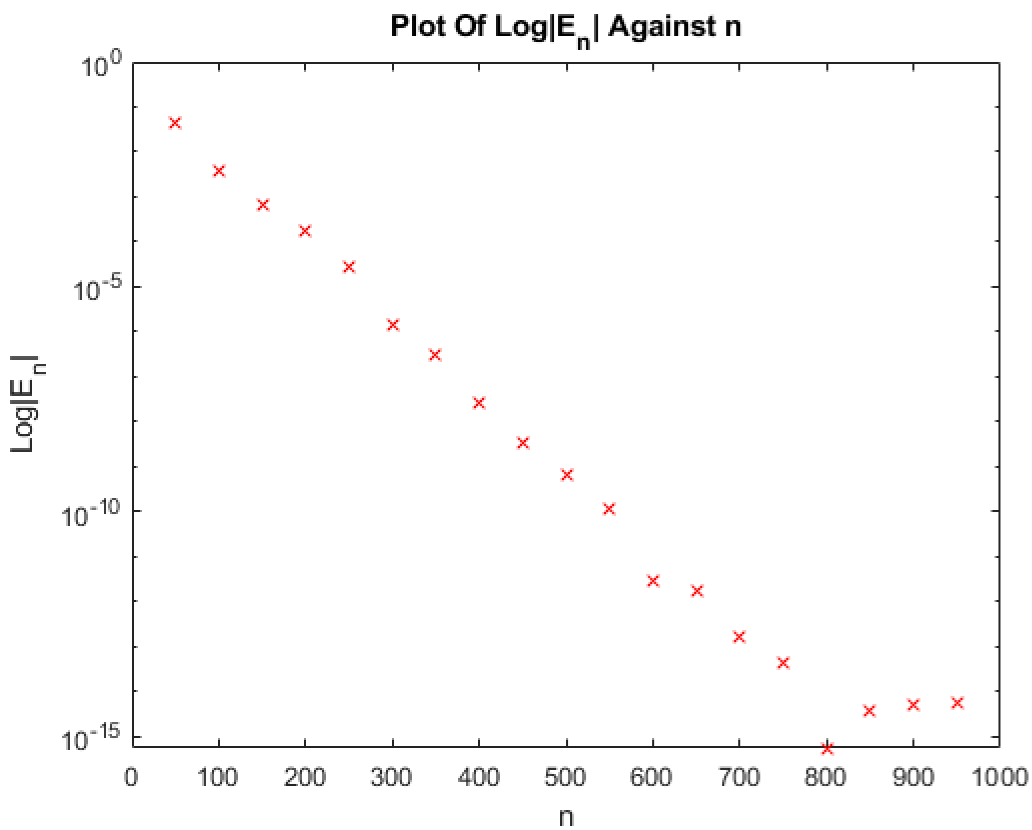
As we can see this function produced the same weights and nodes.

Exercise 3

In this next exercise I will plot $f(x)$ for $x \in [-1, 1]$. I will then make a semilogy plot of $|E_n|$ against n for $n = 50, 100, 150, ..., 950$. However, for this I will take I_{1000} to be I.

```
syms x
f = exp(x)*tanh(4*cos(20*x));
figure(1)
fplot(f,[-1,1])
title('Graph Of f(x)')
xlabel('x')
ylabel('f(x)')
semilogyplot()
```


Graph Of f(x)
f(x)
x
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


Plot Of Log|E_n| Against n
Log|E_n|
n
10^0
10^-5
10^-10
10^-15
0
100
200
300
400
500
600
700
800
900
1000

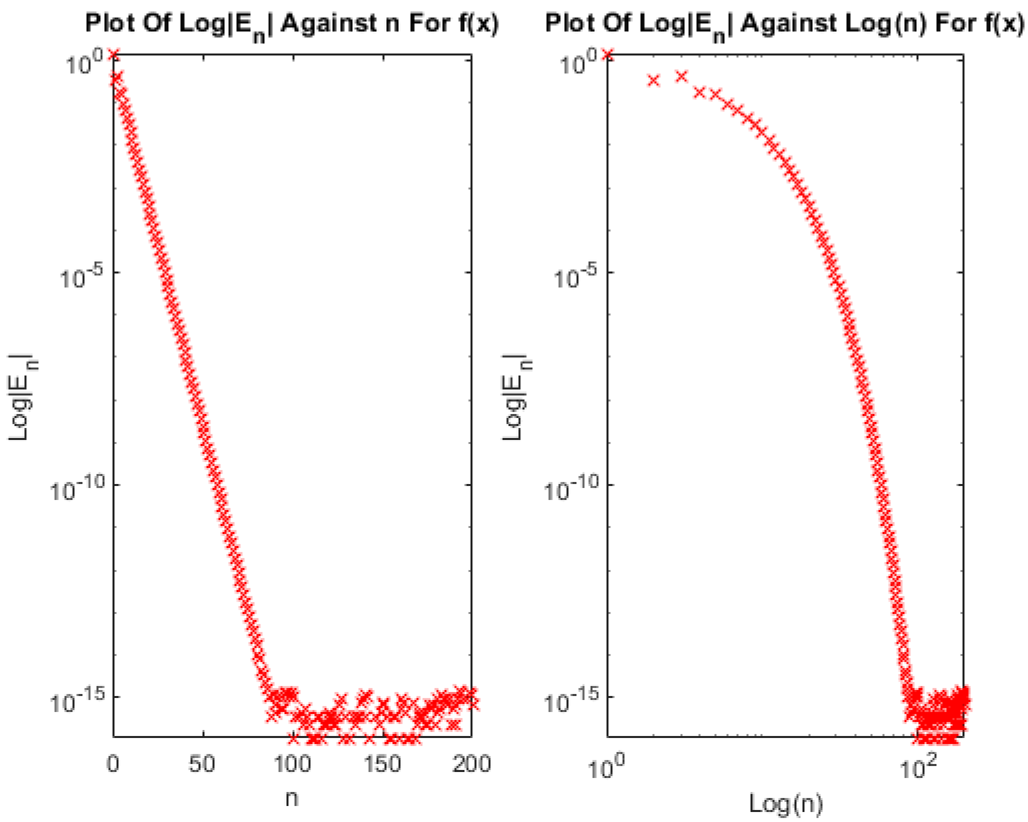
By looking at the semilogy plot we can see that the absolute error decreases exponentially against n initially. It reaches it's lowest point at $n = 800$. However, after this point the error begins to increase slightly, but remains minimal.

Exercise 4

In this exercise I will plot semilogy plots and loglog plots of $|E_n|$ against n for three different functions.

The integral of $f(x)$ for $x \in [-1, 1]$ is $2arctan(5)/5$.

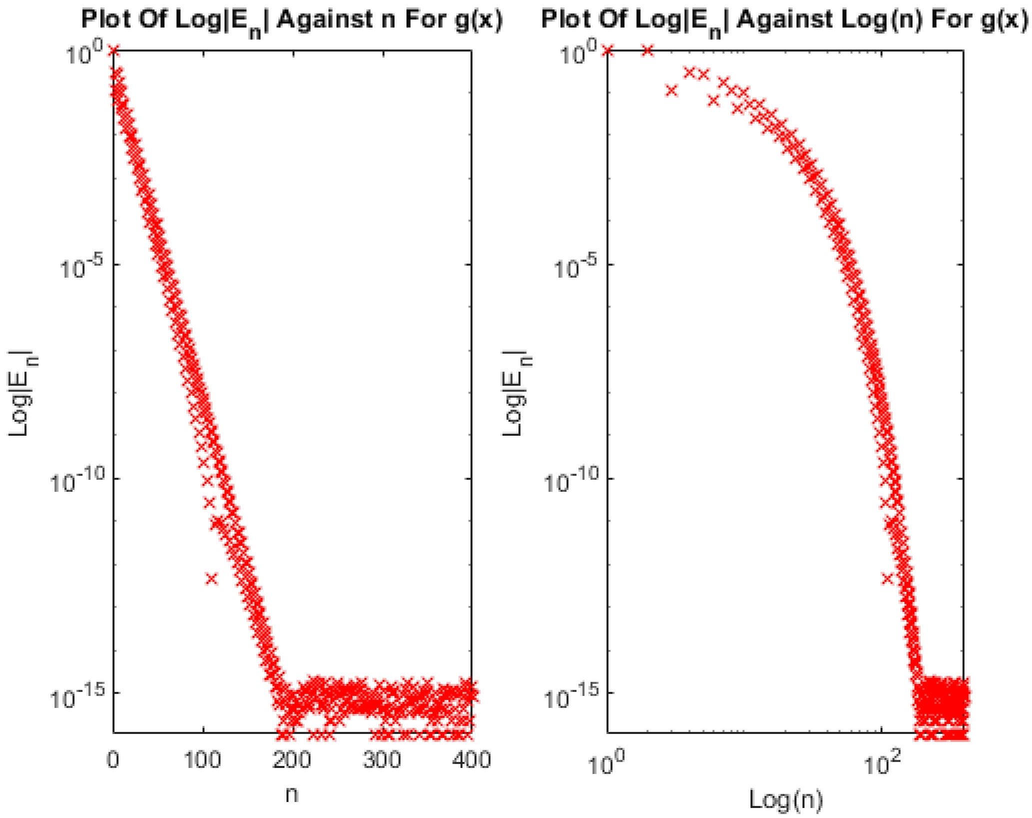
```
fxplots()
```



In this case $n \geq 30$ gives I_n 6 digit accuracy. The shape of these graphs indicate that as n increases, $|E_n|$ decreases exponentially up to a point and then $|E_n|$ remains at a very low value. This shows that $I_n = I$ as $n \to \infty$.

The integral of $g(x)$ for $x \in [-1, 1]$ is $(log(cosh(30)) - log(cosh(10)))/20$.

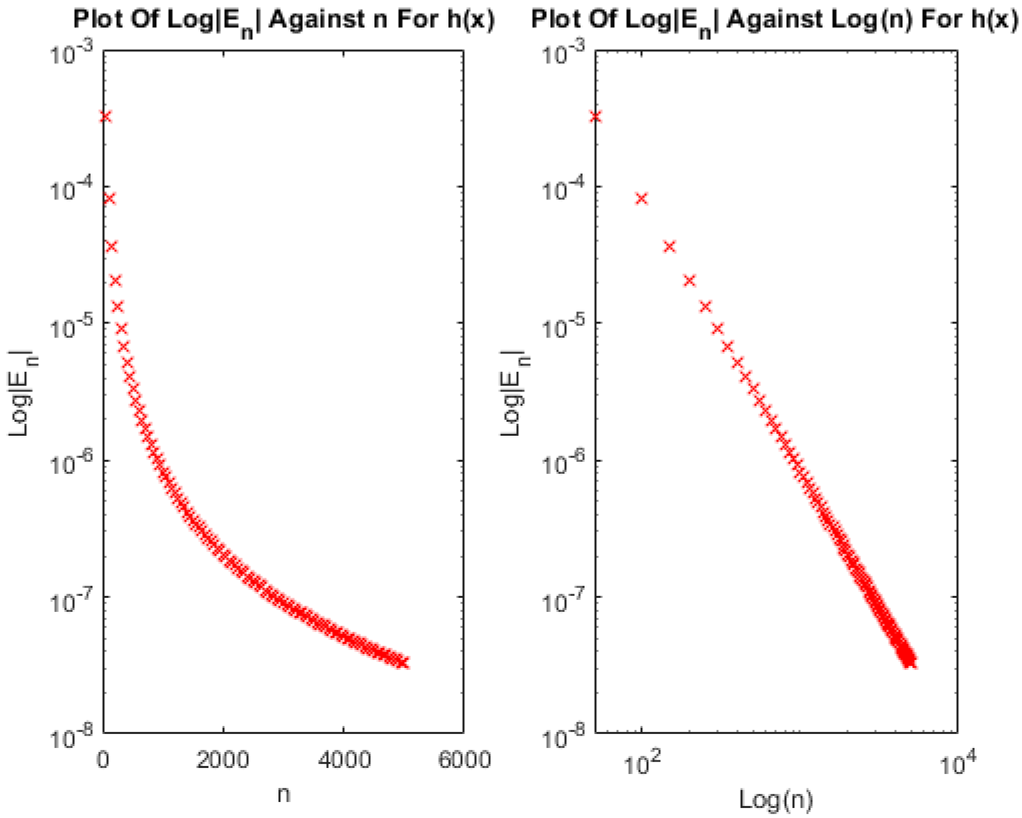
```
gxplots()
```



In this case $n \geq 60$ gives I_n 6 digit accuracy. The shape of these graphs indicate that as n increases, $|E_n|$ decreases exponentially up to a point and then $|E_n|$ remains at a very low value. This shows that $I_n = I$ as $n \to \infty$. The quadrature formula for this integral converged slower than the previous example.

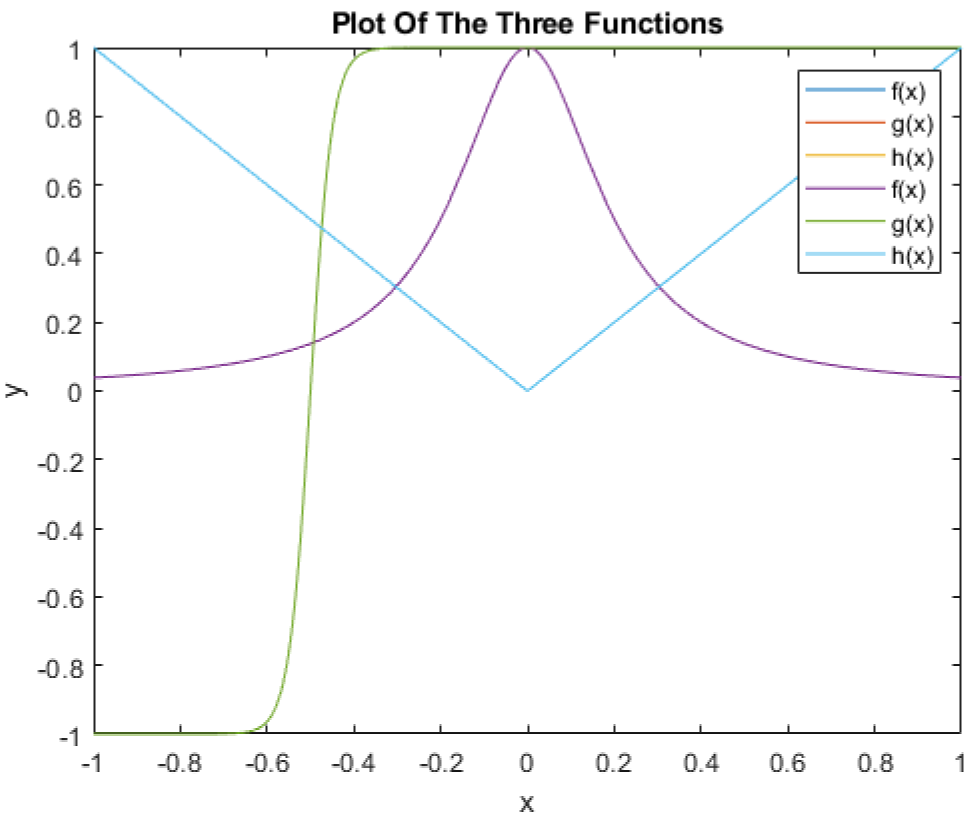
The integral of $h(x)$ for $x \in [-1, 1]$ is 1.

```
hxplots()
```



In this case $n \geq 500$ gives I_n 6 digit accuracy. The shape of these graphs indicate that as n increases, $|E_n|$ decreases linearly. An interesting thing to note is that analytically, this integral is a lot easier to calculate than the other two, however our formula converges at a much slower rate. We can determine why this is by plotting the graphs.

```
syms x
f = 1/(1+25*x^2);
g = tanh(20*(x+0.5));
h = abs(x);
figure(6)
fplot(f,[-1,1],'DisplayName','f(x)')
hold on
fplot(g,[-1,1],'DisplayName','g(x)')
fplot(h,[-1,1],'DisplayName','h(x)')
legend()
title('Plot Of The Three Functions')
xlabel('x')
ylabel('y')
```



We can see that $f(x)$ is smoothest on this interval then $g(x)$ and finally $h(x)$. This explains why quadrature formula converged quickest in that order. This is because Guassian quadrature integrates polynamials up to a given degree exactly, so functions that are closest to a polynomial will converge quicker.

Description Of Functions

This section shows the code written to create the functions that I have used to complete the exercises.

```
type threepointformula.m
type threepointerror.m
type guassq.m
type hyperbolicintegral.m
type semilogyplot.m
type fxintegral.m
type fxplots.m
type gxintegral.m
type gxplots.m
type hxintegral.m
type hxplots.m
```

```
% This function approximates the integral x^k from [-1,1] numerically using
% a quadrature formula. It takes an integer as its argument to decide what
% the function that we are integrating over is.
function in = threepointformula(n)
in = 0; % Initialises value of the integral
```

```
for i = 1:3 % For loop to go through each of the nodes
    if i == 1 % To check which node and weight to use
        in = in + (5/9)*(-(3/5)^0.5)^n; % Updates the integral
    elseif i == 2
        in = in + (8/9)*((0)^0.5)^n;
    else
        in = in + (5/9)*((3/5)^0.5)^n;
    end
end
end


% This function calculates the error of our approximated integral for x^k,
% where k is an integer. It does this for k in [0,10].
function threepointerror()
syms x
for i = 0:10 % For loop to work out the error for each x^k
    integral = int(x^i,x,-1,1); % Actual value of the integral
    approxintegral = threepointformula(i); % Numerical approximation
    en = approxintegral - integral; % Calculates the error
    count = num2str(i); % Used to show for which value of k
    en2str = num2str(double(en));
    X = ['The error when k = ',count,' is ',en2str];
    disp(X)
end
end

% This function takes an integer and produces n distint nodes in [-1,1] and
% its corresponding set of weights. It outputs this as a nx2 vector with
% the nodes in the first column and the weights in the second column.
function [x,w] = guassq(n)
A = zeros(n,n); % Generates the nxn zero matrix
for i = 1:n % Nested for loop to fill in the entries for the matrix
    for j = 1:n
        if abs(j-i) == 1
            % This checks aij is in the upper or lower diagonal position of
            % the matrix
            index = max(i,j);
            % Needs the highest value to correctly work out the value in
            % that position
            A(i,j) = (index-1)/((2*index-3)*(2*index-1))^0.5;
            % Formula for the value of the matrix at that position
        end
    end
end
[V,D] = eig(A);
% Creates a nxn matrix V where the columns are the normalised eigenvalues
% of the tridiagonal symmetric Jacobi matrix. Creates a nxn matrix where
% the diagonal of D are the corresponding eigenvalues.
x = diag(D); % Puts the eigenvalues in a nx1 matrix
w = transpose(2*V(1,:).^2);
% Puts twice the square of the first component of the corresponding
% eigenvector in a nx1 matrix
end
```

```
% This function approximates numerically the integral of e^xtanh(4cos(20x))
% in [-1,1] using the quadrature formula. It takes an integer n to decide how
 many
% nodes in [-1,1] to use and its corresponding weights.
function i = hyperbolicintegral(n)
i = 0;
fx = @(x) exp(x)*tanh(4*cos(20*x));
% Anonymous function that this function is integrating over
[x,w] = guassq(n);
% Uses the guassq function to generate the corresponding nodes and weights
% in a nx2 matrix that is required to approximate the integral
for j = 1:n
    i = i + w(j)*fx(x(j));
    % Formula for quadrature formula
    % w(j) is the weight and x(j) is the node
end
end


% This function makes a semilogy plot of |En| against n for n = 50, 100,
% 150, ..., 950. It uses the numerically approximated integral with n =
% 1000 as a substitute for the actual value of the integral from [-1,1].
function semilogyplot
trueintegral = hyperbolicintegral(1000); % Actual value of the integral
for i = 50:50:950
en = abs(trueintegral - hyperbolicintegral(i));
% The error for each i
figure(2)
semilogy(i,en,'x','Color','red') % Plots the point
hold on
end
title('Plot Of Log|E_n| Against n')
xlabel('n')
ylabel('Log|E_n|')
% Appropriate labels
end

% This function numerically approximates f(x) over [-1,1] using n nodes and
% weights.
function i = fxintegral(n)
i = 0;
fx = @(x) 1/(1+25*x.^2);
[x,w] = guassq(n);
for j = 1:n
    i = i + w(j)*fx(x(j));
end
end

% This function plots |En| over n for f(x) on two subplots. One is a
% semilogy plot and the other is a loglog plot.
function fxplots()
trueintegral = (2*atan(5))/5; % Actual value of the integral
for i = 1:200
```

```
en = abs(trueintegral - fxintegral(i));
figure(3)
subplot(1,2,1)
semilogy(i,en,'x','Color','red')
% Semilogy plot
hold on
subplot(1,2,2)
loglog(i,en,'x','Color','red')
% Loglog plot
hold on
end
subplot(1,2,1)
title('Plot Of Log|E_n| Against n For f(x)')
xlabel('n')
ylabel('Log|E_n|')
subplot(1,2,2)
title('Plot Of Log|E_n| Against Log(n) For f(x)')
xlabel('Log(n)')
ylabel('Log|E_n|')
% Appropriate labels
end

% This function numerically approximates g(x) over [-1,1] using n nodes and
% weights.
function i = gxintegral(n)
i = 0;
fx = @(x) tanh(20*(x+0.5));
[x,w] = guassq(n);
for j = 1:n
    i = i + w(j)*fx(x(j));
end
end

% This function plots |En| over n for g(x) on two subplots. One is a
% semilogy plot and the other is a loglog plot.
function gxplots()
trueintegral = (log(cosh(30))-log(cosh(10)))/20;
% Actual value of the integral
for i = 1:400
en = abs(trueintegral - gxintegral(i));
figure(4)
subplot(1,2,1)
semilogy(i,en,'x','Color','red')
% Semilogy plot
hold on
subplot(1,2,2)
loglog(i,en,'x','Color','red')
% Loglog plot
hold on
end
subplot(1,2,1)
title('Plot Of Log|E_n| Against n For g(x)')
xlabel('n')
ylabel('Log|E_n|')
```

```
subplot(1,2,2)
title('Plot Of Log|E_n| Against Log(n) For g(x)')
xlabel('Log(n)')
ylabel('Log|E_n|')
% Appropriate labels
end

% This function numerically approximates g(x) over [-1,1] using n nodes and
% weights.
function i = hxintegral(n)
i = 0;
fx = @(x) abs(x);
[x,w] = guassq(n);
for j = 1:n
    i = i + w(j)*fx(x(j));
end
end

% This function plots |En| over n for g(x) on two subplots. One is a
% semilogy plot and the other is a loglog plot.
function hxplots()
trueintegral = 1;
for i = 50:50:5000
en = abs(trueintegral - hxintegral(i));
figure(5)
subplot(1,2,1)
semilogy(i,en,'x','Color','red')
% Semilogy plot
hold on
subplot(1,2,2)
loglog(i,en,'x','Color','red')
% Loglog plot
hold on
end
subplot(1,2,1)
title('Plot Of Log|E_n| Against n For h(x)')
xlabel('n')
ylabel('Log|E_n|')
subplot(1,2,2)
title('Plot Of Log|E_n| Against Log(n) For h(x)')
xlabel('Log(n)')
ylabel('Log|E_n|')
% Appropriate labels
end
```

Published with MATLAB® R2021b