Project B - Quadratures

Table of Contents

Candidate number: 1075844

Exercise B1

```
% Initialise the workspace and clear existing figures.
clear
clf

type evaluate_integral.m

% Use the pre-defined function 'evaluate_integral(n)' to calculate the
% exact values of I(f), where f(x)=x^k, k = 0,1,2,3,4,5,6,7 by letting
% n = 4

% We take k = 6,7 because we wish to check the quadrature rule does not
% work for k > 5 if n = 3.

y = evaluate_integral(4);


% Now we are going to verify the quadrature rule when n = 3 with given
% nodes and weights.

% Initialise an empty array z where estimates using the quadrature rule
% will be stored
z = [];

% Define one array for nodes, and one for weights
node = [-sqrt(3/2), 0, sqrt(3/2)];
weight = [sqrt(pi)/6, 2*sqrt(pi)/3, sqrt(pi)/6];

% Use a 'for' loop to iterate through all values of k.
for i = 1:8 % due to indexing issue, i = k+1, so i is from 1 to 8
    z(i)= 0; % initialise the value for I_3(x^(k)), the ith elememt in z
    % another 'for' loop to sum up (w_j)*f(x_j), 1 <= j <= n = 3
    for j = 1:3
        z(i) = z(i) + weight(j)*node(j)^(i-1);
    end
end

% Find and display the differences between exact values and estimates for
```

```
% k = 0,1,...5,6,7
difference = y-z

% Hence we verified that the quadrature rule works for k = 0,1,...,5
% as the differences between exact values and estimates are negligible.

% But the rule doesn't work for k = 6 because the difference between the
% exact value and the estimate is 1.3293, which is significant.


% This function takes the input of an integer 'n' and ouputs a 1-by-2n
% array 'y'.
% The kth element in the array y is equal to the value of the integral in
% (4.1) with f(x) = x^k, where k = 0,1, 2, ..., 2n-1, using (4.3)

function y = evaluate_integral(n)

% Initialise an empty array y.
y = zeros(1,2*n);

% Use a 'for' loop to iterate through all even values of k
for i = 0:n-1 % because 0 is the first even number, and 2n-2 is the last
    k = 2*i; % correspond to the case when k is even
    y(k+1) = (factorial(k)*sqrt(pi))/(2^(k)*factorial(k/2));
    % we need to index y(k+1) because when y(1) is the case when k = 0
end

% We do not need to calculate the case when k is odd because the integral
% equls 0 and I've already created a 1-by-2n zero array.

difference =

  Columns 1 through 7

    0.0000         0    0.0000         0    0.0000         0    1.3293

  Column 8

         0
```

Exercise B2

```
type gausshermite.m

% Find and display the nodes and weights when n = 3.
[calculated_nodes,calculated_weights] = gausshermite(3)

% And compare them with the values given by the question.
given_nodes = double(node)
given_weights = double(weight)

% A comparison gives us that calculated_nodes = given_nodes, and
% calculated_weights = given_weights.


% This function takes the input of an integer 'n' and ouputs two 1-by-n
% arrays, 'x', the nodes, and 'w', the corresponding weights, using the
% Gauss Hermite rule.

function [x,w] = gausshermite(n)

% Initialise a n-by-n zero matrix T.
T = zeros(n);
```

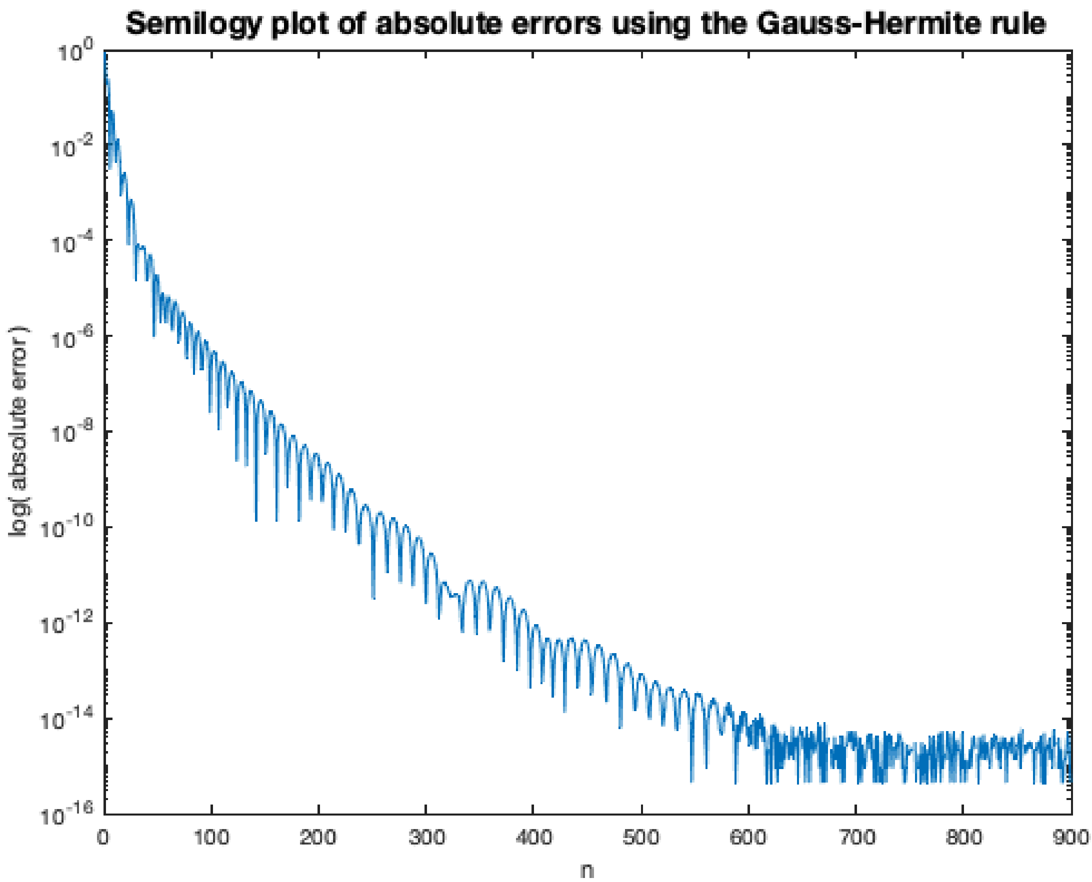
```
% Create the symmetric tridiagonal matrix T
for j = 1:n-1
    T(j,j+1) = sqrt(j/2);
    T(j+1,j) = sqrt(j/2);
end

% Find the eigenvectors, which are the columns of the invertible matrix V,
% and the eigenvalues, which are on the main diagonal of a diagonal
% matrix D.
[V,D] = eig(T);

% Normalise each eigen vector so that all of them have modulus 1.
for i = 1:n
    a = 0;
    for k = 1:n
        a = a + V(k,i)^2; % the squre of modulus of the ith eigenvector
    end
    V(:,i) = V(:,i)/sqrt(a);
end

% Initialise two empty arrays x and w
x = [];
w = [];

% Find the n nodes from eigenvalues and the n weights from the square of
% the first entry of the eigenvectors
for j = 1:n
    x(j) = D(j,j);
    w(j) = (V(1,j))^2 * sqrt(pi);
end


% Note: all numerical values are stored in doubles, as calculations in
% syms take much longer time, rendering the function overly slow for
% Exercise 3.

calculated_nodes =

   -1.2247   -0.0000    1.2247


calculated_weights =

    0.2954    1.1816    0.2954


given_nodes =

   -1.2247         0    1.2247


given_weights =
```

```
    0.2954    1.1816    0.2954
```

Exercise B3

```
% Find the nodes and weights when n = 1000.
[x_1000,w_1000] = gausshermite(1000);

% Initialise an empty array that will be used to store values of
% w_1000(j)*f(x_1000(j))
array = [];

% Calculate w_1000(j)*f(x_1000(j)) for each j and sum them up to get the
% value of I when n = 1000.
for j = 1:1000
    array(j) = w_1000(j)*exp(sin((x_1000(j))^2));
end

% Find and display the approximate when n = 1000.
I_1000 = sum(array)

% Initialise an empty array to store the value of I for different n.
I_n = [];
for n = 1:900
    [x,w] = gausshermite(n); % apply the Gauss Hermite rule for each n.
    new_array = []; % create a new array to store values of w_n(j)*f(x_n(j))
    for j = 1:n
        new_array(j) = w(j)*exp(sin((x(j))^2));
    end
    I_n(n) = abs(sum(new_array)-I_1000);
    % find the absolute differences between I_n and I_1000.
end

% Plot semilogy I_n in figure 1.
figure(1)
semilogy(I_n)
title('Semilogy plot of absolute errors using the Gauss-Hermite
 rule','FontSize',14)
xlabel('n')
ylabel('log( absolute error )')

% The value of absolute eror decreases with increasing n, with small
% oscillations in values. The rate of convergence is fast for n < 600,
% but it becomes slow for n > 600. The value of log(absolute error)
% stablises at around 1e-14.
```

```
I_1000 =

    2.5932
```

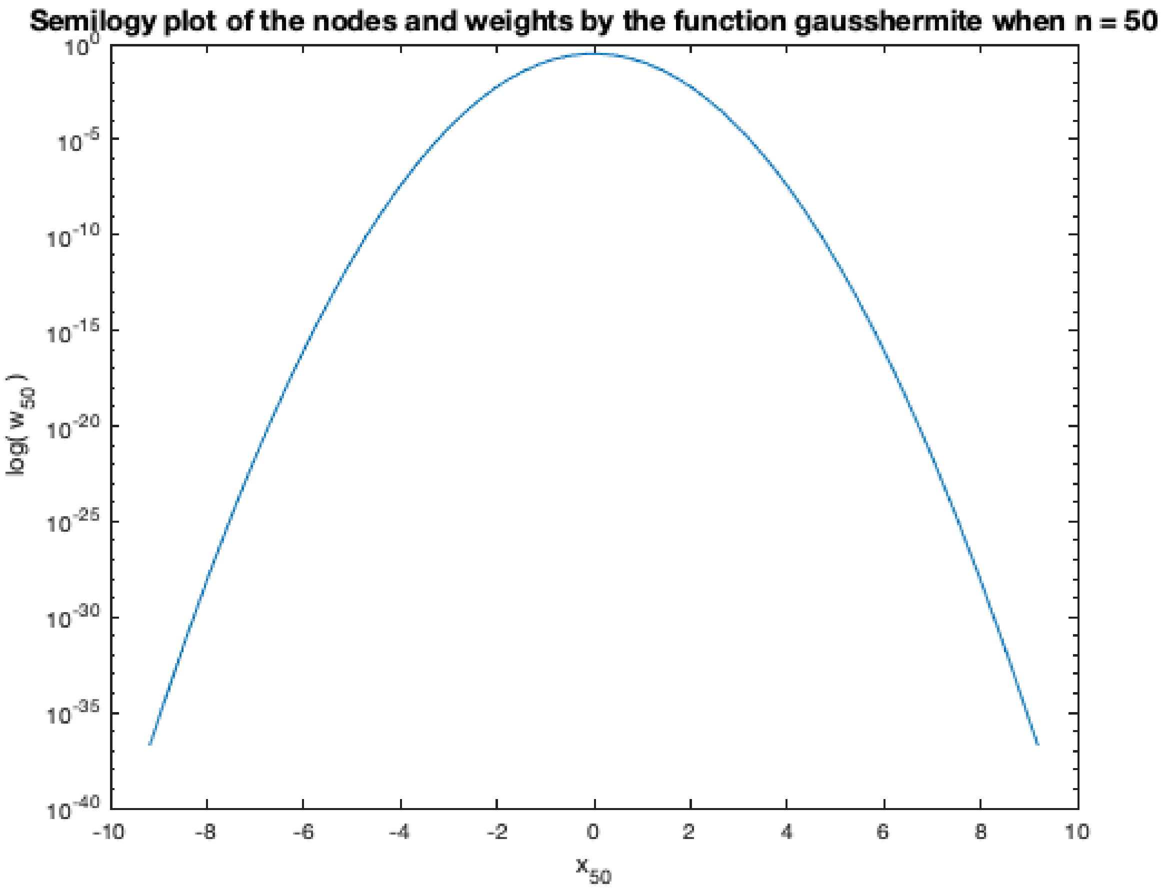


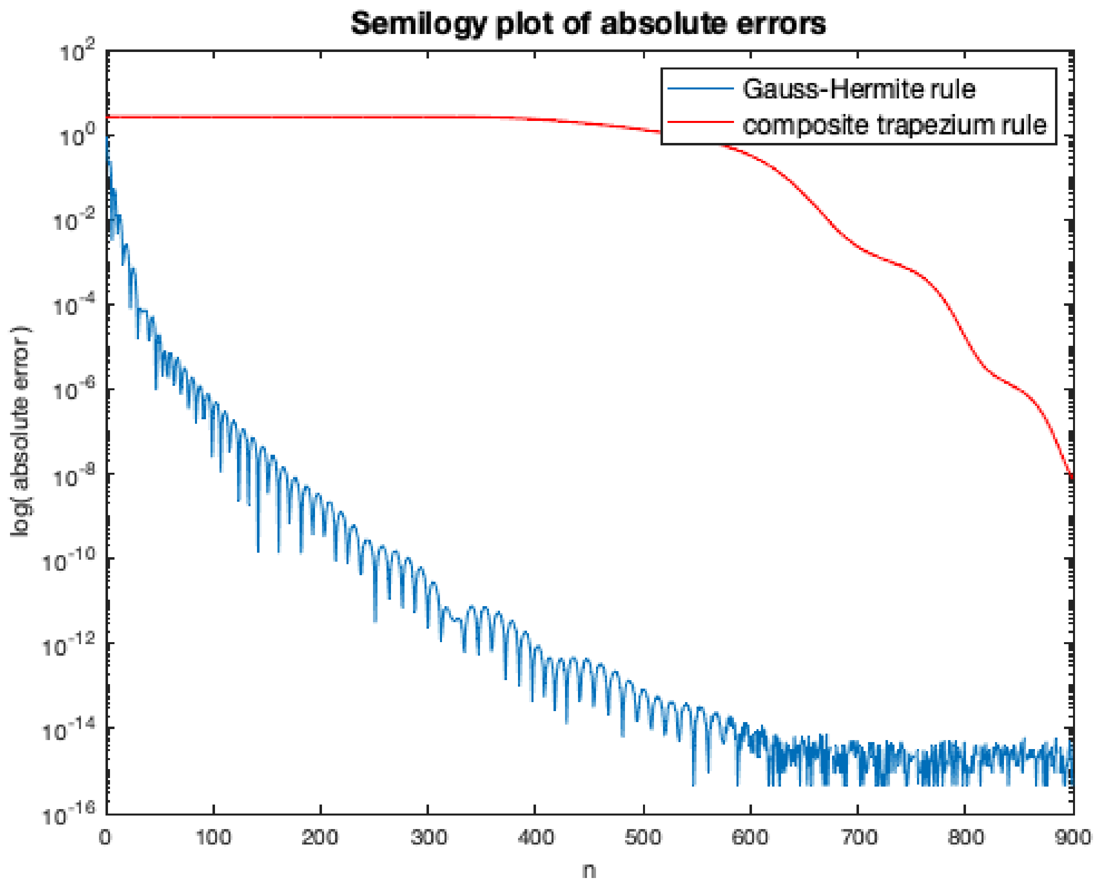
Exercise B4

```
% Find the nodes and weights for n = 50 using the Gauss Hermite rule.
[x_50,w_50] = gausshermite(50);

% Plot semilogy graph for the nodes and weights in figure 2.
figure(2)
semilogy(x_50,w_50)
title('Semilogy plot of the nodes and weights by the function gausshermite
 when n = 50','FontSize',13)
xlabel('x_{50}')
ylabel('log( w_{50} )')

% Hence we note that when |x| > 5 the weights are less than 1e^-10.

type trappts.m

% Use the composite trapisium rule to find nodes and weights for I when
% n = 1000, using nodes in the interval [-5,5] because when |x| > 5 the
% weights are negligible.
[x_trap, w_trap] = trappts(1000,-5,5);
```

```
% Calculate w_trap(j)*exp(-(x_trap(j))^2)*exp(sin((x_trap(j))^2)) for each
% j and sum them up to get the value of I when n = 1000.
array2 = [];
for j = 1:1000
    array2(j) = w_trap(j)*exp(-(x_trap(j))^2+sin((x_trap(j))^2));
end
% Find and display the value of I when n = 1000.
I_trap1000 = sum(array2)

% Initialise an empty array to store the value of I for different n.
I_trapn = [];
for n = 1:900
    [x,w] = trappts(n,-5,5); % apply the composite trapezium rule for each n.
    new_array2 = []; % create a new array to store values
    for j = 1:n
        new_array2(j) = w_trap(j)*exp(-(x_trap(j))^2+sin((x_trap(j))^2));
    end
    I_trapn(n) = abs(sum(new_array2)-I_trap1000);
    % find the absolute differences between I_n and I_1000.
end

% Plot the semilogy for I_n using the composite trapezium rule in figure 1,
% same as the semilogy for I_n using the Gauss Hermite rule.
figure(1)
hold on
semilogy(I_trapn,'r')
title('Semilogy plot of absolute errors','FontSize',14)
legend('Gauss-Hermite rule','composite trapezium rule','FontSize',12)
xlabel('n')
ylabel('log( absolute error )')

% Both approximations give the same value of I_1000 = 2.5932 in 4dp.

% But the approximation using composite trapezium rule is less
% accurate if we look at more decimals than the Gauss Hermite rule
% due to the drop of terms with |x| > 5, though the curve for the
% composite trapezium rule appears to be a lot smoother.

% And approximations using the composite trapezium rule converges very
% slowly when n < 600, but converges faster when n > 600.


% This function takes the input of three integers 'n', 'a,'b' and ouputs
% two 1-by-n arrays, 'x', the nodes, and 'w', the corresponding weights,
% using the composite trapisium rule.

function [x,w] = trappts(n,a,b)

% Initialise an empty array for the nodes.
x = [];
for j = 1:n
    x(j) = a + (j-1)*(b-a)/(n-1); % assign value to each node
end
```

```
% Initialise a array of ones for the weights.
w = ones(1,n);
w = w .* (b-a)/(n-1); % first assign all elements in w the same value
w(1) = (b-a)/(2*(n-1)); % then change w(1) and w(n) to a different value
w(n) = (b-a)/(2*(n-1));

I_trap1000 =

    2.5932
```





Published with MATLAB® R2022b