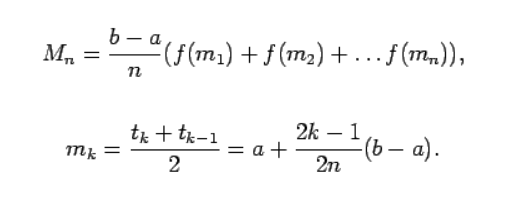
**ECSE 443 - Assignment 4**

**Question 1**

The following function was given to calculate the integral using the various methods.

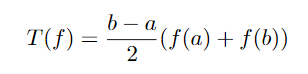
The actual value was found in Matlab to be . For each part we used the relative error as the stopping condition.

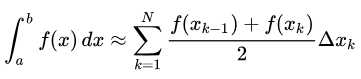
1. Refer to Appendix A, part a for the corresponding Matlab code that was written. The following formula was followed to calculate he area under the curve at each segment using the midpoint rule.



|  |  |
| --- | --- |
| **Number of Segments (Midpoint)** | 9 |
| **I** | 4.355173512185210 |
| **Relative Error** | 3.057463517027002e-07 |

1. Refer to Appendix A, part b for the corresponding Matlab code that was written. The following formula was followed to calculate he area under the curve at each segment using the trapezoidal rule.

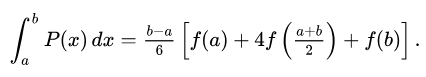




where: 

|  |  |
| --- | --- |
| **Number of Segments (Trapezoidal)** | 9 |
| **I** | 4.355170849024120 |
| **Relative Error** | 3.057475180346966e-07 |

1. Refer to Appendix A, part c for the corresponding Matlab code that was written. The following formula was followed to calculate he area under the curve at each segment using the Simpsons 1/3 rule:



|  |  |
| --- | --- |
| **Number of Segments (Simpsons)** | 8 |
| **I** | 4.355174177932306 |
| **Relative Error** | 4.586099053484477e-07 |

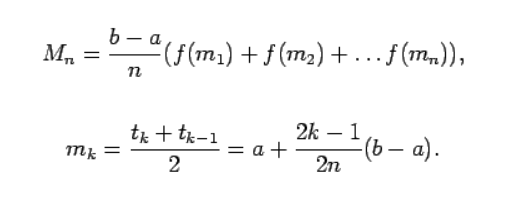
**Question 2**

The following function was given to calculate the integral using the various methods.

The actual value was found in Matlab to be . For each part we used the relative error as the stopping condition.

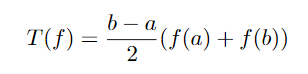
For the following 3 parts the code was written with the same formulas referred to above.

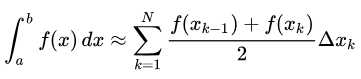
1. Refer to Appendix B, part a for the corresponding Matlab code that was written. The following formula was followed to calculate he area under the curve at each segment using the midpoint rule. To account for the double integral, we repeated the loop to find respect sum under the curve corresponding to each axis. Therefore, the integration was done twice, and a new function was created after the first integration that was used for the second integration.



|  |  |
| --- | --- |
| **Number of Segments (Midpoint)** | 1149 |
| **I** | 790.53561436406453938121866381786 |
| **Absolute Error** | 9.992207839443421e-05 |

1. Refer to Appendix b, part b for the corresponding Matlab code that was written. The following formula was followed to calculate he area under the curve at each segment using the trapezoidal rule. To account for the double integral, we repeated the loop to find respect sum under the curve corresponding to each axis. Therefore, the integration was done twice, and a new function was created after the first integration that was used for the second integration.

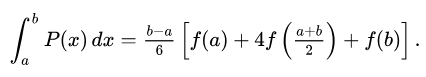




where: 

|  |  |
| --- | --- |
| **Number of Segments (Trapezoidal)** | 1625 |
| **I** | 790.53581419892824806387923450047 |
| **Absolute Error** | 9.991278531424845e-05 |

1. Refer to Appendix B, part c for the corresponding Matlab code that was written. The following formula was followed to calculate he area under the curve at each segment using the Simpsons 1/3 rule. To account for the double integral, we repeated the loop to find respect sum under the curve corresponding to each axis. Therefore, the integration was done twice, and a new function was created after the first integration that was used for the second integration.



|  |  |
| --- | --- |
| **Number of Segments (Simpsons)** | 12 |
| **I** | 790.53580068440866840992226794696 |
| **Absolute Error** | 8.639826573459449e-05 |

**Question 3**

1. To calculate the fifth order backward difference with first order accuracy. We begin with the known function.

The fifth order derivative is calculated by integrating this formula recursively.

1. In order to solve the first derivative function, we begin with the following the function that we will fill its coefficients in using linear algebra.

To solve the function, we will plug in the multiple Taylor series function below.

Then we plug these functions into the original function.

If we go ahead and break the big equation into the respective system of equations for each derivative, we can plug it into a matrix to solve the system of equations.

In Matlab, the coefficient matrix a is found:

A = [1 1 1 1; 0 1 2 3; 0 1/2 2 9/2; 0 1/6 4/3 27/6];

B = [0;1;0;0];

coeffs = inv(A)\*B

coeffs = 4×1

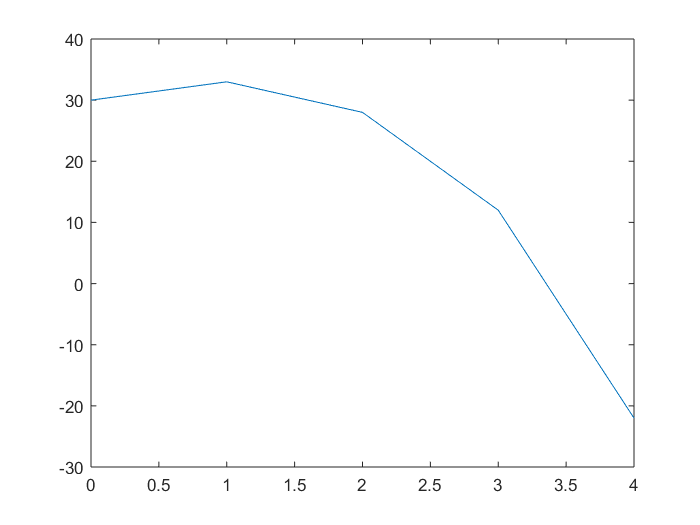
-1.833333333333333  
 3.000000000000000  
 -1.500000000000000  
 0.333333333333333

Therefore, the following equation is found:

**Question 4**

\*\*\*In each of the following examples we only displayed the final equation, as the process to solve for the equation is the same process outlines in **question 3**. Please see above for the functions.

The given points correspond to the plot below:



1. Refer to Appendix C, part a. To calculate , accurate to the second order h2, we are unable to use the central difference method because we do not have the values before x=0. We will use the second order accurate forward difference, which uses the forward difference method with h and 2h points. This takes the Taylor series for the derivative and keeps values up until h2 so we can be second order accurate.

The following is calculated.

1. Refer to Appendix C, part b. To calculate , accurate to the second order h2, we use the central difference method since we have the coordinates around it.

The following is found:

1. Refer to Appendix C, part c. To calculate , accurate to the second order h2, we are unable to use the central difference method because we do not have the values before x=0. We will use the second order accurate backward difference, which uses the forward difference method with h and 2h points. This takes the Taylor series for the derivative and keeps values up until h2 so we can be second order accurate.

The following is calculated.

1. Refer to Appendix C, part d. To calculate , accurate to the second order h2, we are unable to use the central difference method because we do not have the values before x=0. We will use the second order accurate forward difference.

The following is calculated to.

**Appendix**

**Appendix A – Question 1 Matlab Code**

Question 1

syms f(t)

format long

% create the function variable and its integration endpoints

f(t) = log(5-4\*cos(t));

% create a function that can be passed to functions

p = @(t) log(5-4\*cos(t));

a = 0;

b = pi;

% set threshold to the error

thresh = 10^-6;

% actual answer

actual = int(f(t), t, a, b)

actual = C:\Users\bjay2\AppData\Local\Temp\ConnectorClipboard9085067487415196558\image15535653416480.png

actual1 = integral(p, a, b)

actual1 =

4.355172180607203

part a, the midpoint rule

% the number of segments/points that will be used , assume >1

N\_m = 1;

% set the sum of areas to 0

M = 0;

% check if the value vary by a large enough threshold

while(abs(M-actual)/actual > thresh)

% reset the sum of areas to 0

M = 0;

% find the step size

dx = (b-a)/N\_m;

for i=1:N\_m

% calculate the midpoint formula at each segment and sum the area

% under the curve

x = a + 0.5\*(2\*i-1)\*dx;

M = M + f(x)\*dx;

end

% increment the number of segmenets

N\_m = N\_m+1;

end

N\_m = N\_m - 1

N\_m =

9

relativeError = double(abs(M-actual)/actual)

relativeError =

3.057463517027002e-07

double(M)

ans =

4.355173512185210

% using my function

% [M,N\_M] = MidpointIntegration(p,a,b,thresh)

part b, the trapezoid rule

% the number of segments/points that will be used , assume >1

N\_t = 1;

% T starting point to 0

T = 0;

% check if the value vary by a large enough threshold

while(abs(T-actual)/actual > thresh)

% find the step size

dx = (b-a)/N\_t;

% we want x+1 and x for we need to x vectors

x1 = a:dx:b-dx;

x2 = a+dx:dx:b;

% calculate the trapzoid method using vectors

y = f(x2) +f(x1);

T = 0.5\*sum(y\*dx);

% T = 0;

% T = 0.5\*(f(a) + f(b));

% for i=1:N\_t-1

% T = T + f(a + i\*dx);

% end

% T = dx\*T;

% increment the number of segments

N\_t = N\_t+1;

end

N\_t = N\_t - 1

N\_t =

9

relativeError = double(abs(T-actual)/actual)

relativeError =

3.057475180346966e-07

double(T)

ans =

4.355170849024120

part c, the simpsons rule

% the number of segments/points that will be used , assume >1

N\_s = 1;

% set S = 0

S = 0;

% check if the value vary by a large enough threshold

while(abs(S-actual)/actual > thresh)

% reset the sum of of area segments under the curve

S = 0;

% find the step size

dx = (b-a)/N\_s;

for i=1:N\_s

% calculate the integral using the simpsons method

x = a + (i-1)\*dx;

y = (dx/6)\*(f(x) + 4\*f(x+dx/2) + f(x + dx));

S = S + y;

end

% increment the number of segments

N\_s = N\_s + 1;

end

N\_s = N\_s - 1

N\_s =

8

relativeError = double(abs(S-actual)/actual)

relativeError =

4.586099053484477e-07

double(S)

ans =

4.355174177932306

**Appendix B – Question 2 Matlab Code**

Question 2

syms f(s,t) s h(s)

format long

% create the function variable and its integration endpoints

f(s,t) = s^2 + t;

a = 2;

b = 3;

c = s;

d = 2\*s^3;

c\_ = @(x) x;

d\_ = @(x) 2\*x.^3;

% create a function that can be passed to functions

p = @(x,y) x.^2 + y;

% set threshold to the error

thresh = 10^-4;

% actual answer

actual = integral2(p, a, b, c\_, d\_)

actual =

7.905357142861429e+02

f(s,t)

ans = C:\Users\bjay2\AppData\Local\Temp\ConnectorClipboard9085067487415196558\image15535657577750.png

part a, the midpoint rule

% the number of segments/points that will be used , assume >1148

N\_m = 1149; % answer is 1149

% set the sum of areas to 0

M = 0;

% check if the value vary by a large enough threshold

while(abs(M-actual) > thresh)

% reset the sum of areas to 0

M = 0;

temp = 0;

% find the step size

dy = (d-c)/N\_m;

for i=1:N\_m

% calculate the midpoint formula at each segment and sum the area

% under the curve for the first part section under y

y = c + 0.5\*(2\*i-1)\*dy;

temp = temp + f(s,y)\*dy;

end

% find the step size

dx = (b-a)/N\_m;

% keep the function in a symbolic function

h(s) = temp;

% repeat the algorithm for the area under the curve on the other x-axis

for i=1:N\_m

x = a + 0.5\*(2\*i-1)\*dx;

M = M + h(x)\*dx;

end

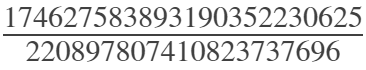
% increment the number of segmenets

N\_m = N\_m+1;

end

M

M =



N\_m = N\_m - 1

N\_m =

1149

absoluteError = double(abs(M-actual))

absoluteError =

9.992207839443421e-05

part b, the trapezoidal rule

% the number of segments/points that will be used , assume >1624

N\_t = 1625; % actual answer is 1625

% T starting point to 0

T = 0;

% check if the value vary by a large enough threshold

while(abs(T-actual) > thresh)

% find the step size corresponding to the y-axis

dy = (d-c)/N\_t;

% compute the area under the curve of the y-axis

temp = 0.5\*(f(s,c) + f(s,d));

for i=1:N\_t-1

temp = temp + f(s,c + i\*dy);

end

temp = dy\*temp;

% find the step size corresponding to the x-axis

dx = (b-a)/N\_t;

h(s)= temp;

% compute the area under the curve of the x-axis

T = 0.5\*(h(a) + h(b));

for i=1:N\_t-1

T= T + h(a + i\*dx);

end

T = dx\*T;

% increment the number of segments

N\_t = N\_t+1;

end

N\_t = N\_t - 1

N\_t =

1625

T

T =



absoluteError = double(abs(T-actual))

absoluteError =

9.991278531424845e-05

part c, the simpsons rule

% the number of segments/points that will be used , assume >1

N\_s = 1;

% set S = 0

S = 0;

% check if the value vary by a large enough threshold

while(abs(S-actual) > thresh)

% reset the sum of of area segments under the curve

S = 0;

temp = 0;

% find the step size on the y-axis

dy = (d-c)/N\_s;

% calculate the integral using the simpsons method for hte y-axis

for i=1:N\_s

y = c + (i-1)\*dy;

temp = temp + (dy/6)\*(f(s,y) + 4\*f(s,y+dy/2) + f(s,y+dy));

end

h(s) = temp;

% find the step size on the x-axis

dx = (b-a)/N\_s;

% calculate the integral using the simpsons method for x-axis

for i=1:N\_s

x = a + (i-1)\*dx;

S = S + (dx/6)\*(h(x) + 4\*h(x+dx/2) + h(x+dx));

end

% increment the number of segments

N\_s = N\_s + 1;

end

N\_s = N\_s - 1

N\_s =

12

S

S =



absoluteError = double(abs(S-actual))

absoluteError =

8.639826573459449e-05

**Appendix C – Question 4 Matlab Code**

Question 4

format long

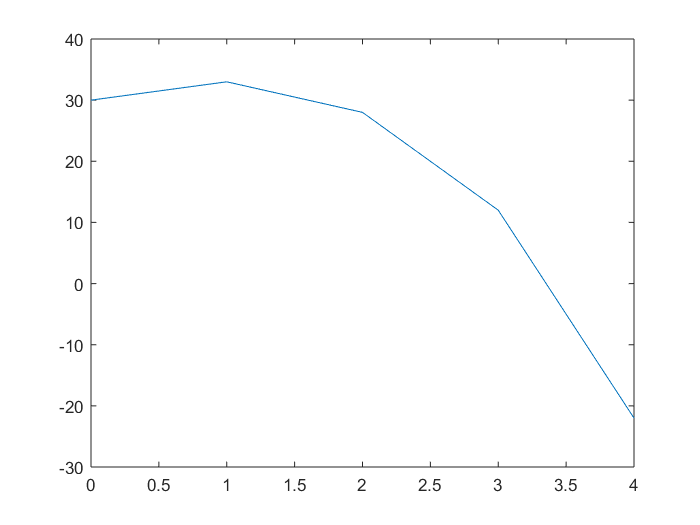
% import the data to be used

data = importdata('Ass\_4\_data\_functQ4.txt');

x = data(:,1);

y = data(:,2);

plot(x,y)



% find the step size

h = x(2)-x(1)

h =

1

Calculate f'(0)

% the point is zero, but to plug into the function, the array data starts

% at index 1

x\_0 = 0+1;

% find the second order forward approximation

fd\_0 = (-y(x\_0 + 2\*h) + 4\*y(x\_0 + h) - 3\*y(x\_0))/(2\*h)

fd\_0 =

7

Calculate f'(2)

% the point is 2, but to plug into the function, the array data starts

% at index 1

x\_1 = 2+1;

% find the second order centered approximation

cd\_1 = (y(x\_1 + h) - y(x\_1 - h))/(2\*h)

cd\_1 =

-10.500000000000000

Calculate f'(4)

% the point is 4, but to plug into the function, the array data starts

% at index 1

x\_2 = 4+1;

% find the second order backward approximation

bd\_2 = (y(x\_2 - 2\*h) - 4\*y(x\_2 - h) + 3\*y(x\_2))/(2\*h)

bd\_2 =

-43

Calculate f''(0)

% the point is zero, but to plug into the function, the array data starts

% at index 1

x\_3 = 0+1;

% find the second order forward approximation

fd\_3 = (-y(x\_3 + 3\*h) + 4\*y(x\_3 + 2\*h) - 5\*y(x\_3 + h) + 2\*y(x\_3))/h^2

fd\_3 =

-5