Ezz Aboulezz

260677463

ECSE 443 Final Project

Q1.

In question 1 we compute the integral using Simpson’s rule. The value computed for the integral was 0.785325349762923. This answer matches the true value within 4 decimals.

Q2.

In question 2 we compute the integral using the midpoint method. The value computed for the integral was 0.623775746323755.

Q3. (unsolved)

Q4.

In question 4 the values of the coefficients were determined using forward finite difference. We assumed y(a) => y(2), y(b) => y(3), and y(c) => y(4)

The coefficients were found to be:

1.0e+02 \* -0.139547265702589

1.0e+02 \* -0.861509728326910

1.0e+02 \* -2.369463437976441

Q5.

In question 5 we compute the answer and the square error of the solution using LU factorization.

The solution was: 18.000000000000000

25.999999999999986

33.999999999999901

82.000000000000000

And the square error was 2.524354896707238e-27. This was very reasonable.

Q6.

In question 6 we compute the answer and the square error of the solution using fixed point iteration.

The solution was: 18.000000000000000

26.000000000000004

33.999999999999993

82.000000000000000

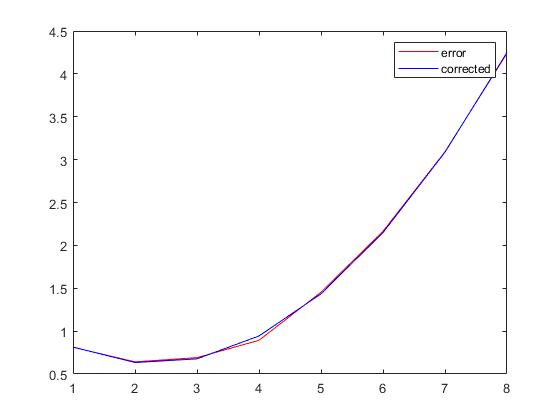
And the square error was 1.577721810442024e-29. This was very reasonable.

Q7.

In question 7 we find the smallest possible root using the secant method. The root was found to be 4.730040769668035.

Q8.

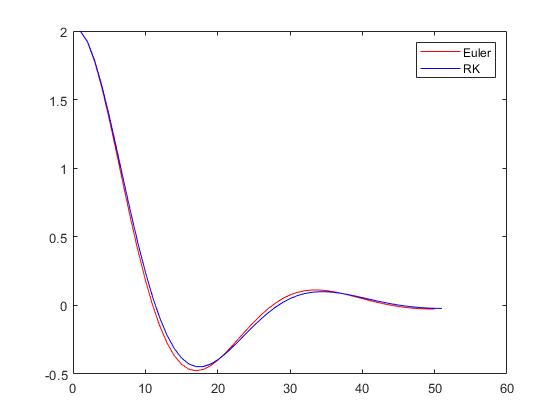
In this question the 4th point was found to be erroneous and both the error and the corrected graphs were plotted. The point was corrected to 0.943642857142857.



Q9.

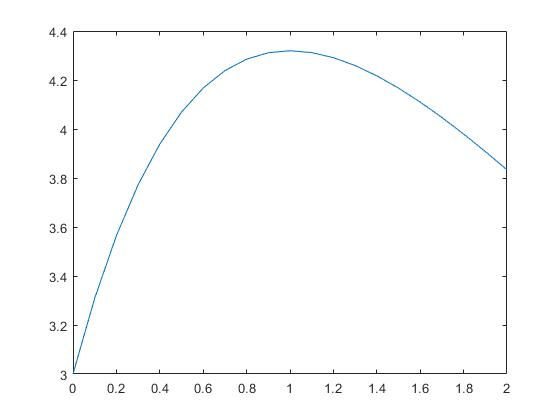
In this question both Euler and RK methods were used and the results were plotted.

The resulting plots were very close to being identical.



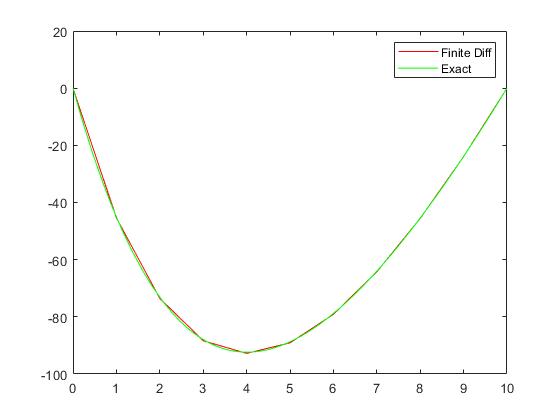
Q10.

In this question the second order RK was used to solve the ODE and plot the result.



Q11.

In this question we solve the boundary value problem using finite difference method. Both the solution and the exact graphs were plotted.



Q12. (unsolved)

Appendix: Code used for all questions

Q1

%function, limits and true value

format long;

syms x;

f = @(x) 1/(exp(x) + exp(-x));

LL = 0;

UL = inf;

trueValue = int(f(x), LL, UL);

%number of steps, step size, accuracy, relative error and sum

n = 0;

s = 0;

acc = 0.0001;

relativeError = 100;

simSum = 0;

while(relativeError > acc)

%large upper limit = 100

s = (100 - LL)/n;

simSum = 0;

for i=1:n

%simpson's rule

add = (s/6)\*(f(LL + (i-1)\*s) + f((LL + (i-1)\*s) + s) + 4\*f((LL + (i-1)\*s) + s/2));

simSum = simSum + add;

end

relativeError = abs(simSum - trueValue)/trueValue;

n = n + 1;

end

display(simSum);

display(trueValue);

Q2

%limits, step size, number of steps and sum

format long;

LL = 0;

UL = 1;

s = 0.01;

n = (UL - LL)/s;

midSum = 0;

x = 0;

deltaX = (UL - LL)/100;

for i=1:n %100

%midpoint rule

x = LL + (i\*(deltaX)) - ((deltaX)/2);

%function

y = (x^3)/((1 - x^2)^0.5);

midSum = midSum + y\*deltaX;

end

display(midSum);

Q3 (unsolved)

Q4

%output and step size

format long;

y = [1, 0.1160, -0.1084, -0.0409, 0.0052, 0.0067, 0.009];

s = 0.25;

%get matrix -> [y''(a), y'(a), y(a);

%y''(b), y'(b), y(b);

%y''(c), y'(c), y(c)]

%y(a) => y(2), y(b) => y(3), and y(c) => y(4)

yA = y(2);

yB = y(3);

yC = y(4);

%forward finite difference

%for y''(a), y'(a)

y1stDerA = ((-11/6)\*y(2) + 3\*y(3) - 1.5\*y(4) + (1/3)\*y(5))/s;

y2ndDerA = (2\*y(2) - 5\*y(3) + 4\*y(4) - y(5))/(s\*s);

%for y''(b), y'(b)

y1stDerB = ((-11/6)\*y(3) + 3\*y(4) - 1.5\*y(5) + (1/3)\*y(6))/s;

y2ndDerB = (2\*y(3) - 5\*y(4) + 4\*y(5) - y(6))/(s\*s);

%for y''(c), y'(c)

y1stDerC = ((-11/6)\*y(4) + 3\*y(5) - 1.5\*y(6) + (1/3)\*y(7))/s;

y2ndDerC = (2\*y(4) - 5\*y(5) + 4\*y(6) - y(7))/(s\*s);

%assemble answer

matrix = [y2ndDerA, y1stDerA, yA;

y2ndDerB, y1stDerB, yB;

y2ndDerC, y1stDerC, yC];

answer = matrix\[1, 1, 1]';

display(answer);

Q5

%matrix A, vector b and number of rows

A = [3 -5 47 20; 11 16 17 10; 56 22 11 -18; 17 66 -12 7];

b = [18; 26; 34; 82;];

r = 4;

%U and L

U = A;

L = eye(4);

%doolittle algorithm to get U & L

for i = 1:r-1

for j = i+1:r

L(j,i) = U(j,i)/U(i,i);

U(j,i:r) = U(j,i:r) - (U(i,i:r)\*L(j,i));

end

end

%solve for y using y = b/L

y = L^(-1)\*b;

%solve for x using x = y/U

x = U^(-1)\*y;

%verify

s = zeros(4,1);

for i=1:4

s(i,1) = A(i,1)\*x(1,1) + A(i,2)\*x(2,1) + A(i,3)\*x(3,1) + A(i,4)\*x(4,1);

end

%get square error

squareError = 0;

e = zeros(4,1);

for i=1:4

e(i,1) = (b(i,1) - s(i,1))^2;

end

for i=1:4

squareError = squareError + e(i,1);

end

squareError = squareError/4;

display(squareError);

disp(s);

Q6

%matrix A, vector b and x (Ax = b)

format long;

A = [3 -5 47 20; 11 16 17 10; 56 22 11 -18; 17 66 -12 7];

b = [18; 26; 34; 82;];

x = [0; 0; 0; 0];

for i=1:1000 %large number to decrease error

x(4) = (b(2) - A(2)\*x(1) - A(2,2)\*x(2) - A(2,3)\*x(3))/A(2,4);

x(3) = (b(1) - A(1)\*x(1) - A(1,2)\*x(2) - A(1,4)\*x(4))/A(1,3);

x(2) = (b(4) - A(4)\*x(1) - A(4,3)\*x(3) - A(4,4)\*x(4))/A(4,2);

x(1) = (b(3) - A(3,2)\*x(2) - A(3,3)\*x(3) - A(3,4)\*x(4))/A(3);

x = x';

end

%verify

s = zeros(4,1);

for i=1:4

s(i,1) = A(i,1)\*x(1,1) + A(i,2)\*x(2,1) + A(i,3)\*x(3,1) + A(i,4)\*x(4,1);

end

%get square error

squareError = 0;

e = zeros(4,1);

for i=1:4

e(i,1) = (b(i,1) - s(i,1))^2;

end

for i=1:4

squareError = squareError + e(i,1);

end

squareError = squareError/4;

display(squareError);

disp(s);

Q7

%function, guesses and relative error

syms x;

f = @(x) cos(x)\*cosh(x) - 1;

g1 = 4;

g2 = 6;

relativeError = 10^(-4);

%function at guess

fg1 = f(g1);

deltaX = 100;

%secant method

while(abs(deltaX) > relativeError)

fg2 = f(g2);

deltaX = (g2-g1)\*fg2/(fg2-fg1);

%update

fg1 = fg2;

g1 = g2;

g2 = g2 - deltaX;

end

answer = g2;

display(answer);

Q8

%error at x = 4

%input and output

x = [1, 2, 3, 4, 5, 6, 7, 8];

y = [0.812, 0.642, 0.691, 0.893, 1.454, 2.164, 3.092, 4.24];

%coefficients and corrected function

z = ones(size(x));

A = [z', x', (x.\*x)'];

nA = A'\*A;

nB = A'\*y';

coeff = nA\nB;

correct = coeff(1) + coeff(2).\*x + coeff(3).\*x.^2;

%display corrected point

display(correct(4));

%display both plots

plot(x, y, 'red')

hold on;

plot(x, correct, 'blue')

hold off;

legend('error', 'corrected')

Q9

%initial conditions, t's, step size and number of steps

y0 = 2;

dy0 = 0;

t0 = 0;

t5 = 5;

s = 0.1;

n = (t5 - t0)/s;

%Euler

for i=1:n

%#ok<\*SAGROW>

y(i) = y0 + (dy0\*s);

dy(i)= dy0 + (-2\*dy0 - 4\*y0)\*s;

%update IC's

y0 = y(i);

dy0 = dy(i);

end

%reset initial conditions

y0 = 2;

dy0 = 0;

g = dy0;

f = @(g,y) -2\*g - 4\*y;

%R-K

RK = zeros(1,10);

i = 1;

for t = t0:s:t5

K1 = s\*g;

R1 = s\*f(g, y0);

K2 = s\*(g + K1/2);

R2 = s\*f(g + R1/2, y0 + K1/2);

K3 = s\*(g + K2/2);

R3 = s\*f(g + R2/2, y0 + K2/2);

K4 = s\*(g + K3/2);

R4 = s\*f(g + R3/2, y0 + K3/2);

Ky = (K1 + 2\*K2 + 2\*K3 + K4)/6;

Kg = (R1 + 2\*R2 + 2\*R3 + R4)/6;

y0 = y0 + Ky;

RK(i) = y0;

i = i + 1;

g = g + Kg;

end

plot(y,'red')

hold on

plot(RK,'blue')

hold off

legend('Euler', 'RK')

Q10

%x's, initial condition, step size, number of steps

x0 = 0;

x2 = 2;

y0 = 3;

s = 0.1;

n = (x2-x0)/s;

x = (x0:s:x2)';

y = zeros(10, 1);

%second-order R-K

for i=1:n

y(1) = y0;

y1 = -1.2\*y(i) + 7\*exp(-0.3\*x(i));

y2 = -1.2\*(y(i) + s\*y1) + 7\*exp(-0.3\*(x(i) + s));

y(i+1) = y(i) + (0.5\*s)\*(y1 + y2);

end

plot(x,y);

Q11

syms x

syms y(x)

%exact using dsolve and diff

%function and conditions

fnc = diff(y, x, 2) + (1/4)\*diff(y, x, 1) == 8;

c1 = y(0) == 0;

c2 = y(10) == 0;

exactF = dsolve(fnc, [c1 c2]);

%finite difference method

%step size, t's, t range, number of steps, and equ

eq = 8;

s = 1;

t0 = 0;

t10 = 10;

t = t0:s:t10;

n = 1 + (t10 - t0)/s;

%prepare equations

%yh = y + h, y\_h = y - h

syms y

syms yh

syms y\_h

dydt = (yh - y\_h)/2\*s;

d2ydt2 = (yh - 2\*y + y\_h)/s^2;

%finite difference equation

fnc(y\_h, y, yh) = dydt/4 + d2ydt2;

%set A and B

A(1:n,1:n)=0;

A(1,1)=1;

A(n,n)=1;

%fill A coefficients

for i=2:n-1

A(i,i-1) = fnc(1,0,0);

A(i,i) = fnc(0,1,0);

A(i,i+1) = fnc(0,0,1);

end

B(1:n,1)=0;

B(2:n-1,1)=eq;

finiteD = (A'\*A)\(A'\*B);

plot(t,finiteD,'red');

hold on

fplot(exactF,[0,10],'green')

hold off

legend('Finite Diff', 'Exact')