**MSDM5004 HW2 solution**

**Q1**

First get

**Part (1) – Jacobi method**

1. Compute the coefficient and
2. Iteration by , begin from

**Part (2) – Gauss Siedel**

1. Compute the coefficient and
2. Iteration by , begin from

**Q2**

Note that norm of a vector is

**Part (1) – Gauss Siedel**

A = [4,1,1,-1;

1,4,-1,-1;

1,-1,3,1;

-1,-1,1,5];

b = [-3,-2,2,5].';

D = diag(diag(A));

U = -triu(A,1);

L = -tril(A,-1);

g1 = inv(D-L)\*U;

g2 = inv(D-L)\*b;

esp = 1/1000;

x0 = [0,0,0,1].';

x1 = 100\*ones(4,1);

while norm(x1-x0, Inf) >= esp

x1 = x0;

x0 = g1\*x0 + g2;

end

disp(x0)

**Console display:**

-0.753059796800029

0.040886583057744

0.691516167015457

0.719262123848452

**Part (2) – SOR**

A = [4,1,1,-1;

1,4,-1,-1;

1,-1,3,1;

-1,-1,1,5];

b = [-3,-2,2,5].';

w = 1.2;

D = diag(diag(A));

U = -triu(A,1);

L = -tril(A,-1);

g1 = inv(D-w\*L)\*((1-w)\*D+w\*U);

g2 = w\*inv(D-w\*L)\*b;

esp = 1/1000;

x0 = [0,0,0,1].';

x1 = 100\*ones(4,1);

while norm(x1-x0, Inf) >= esp

x1 = x0;

x0 = g1\*x0 + g2;

end

disp(x0)

**Console display:**

-0.753455643103498

0.041243058029276

0.691714074061100

0.719166490715922

**Q3**

f = @(x) sin(x)/x^2;

h = 1e-2;

x0 = 3;

% Computing derivative

df = @(x,h) (f(x+h)-f(x-h))/2/h;

disp("derivative: " + df(x0,h))

% Checking order of numerical scheme

n = 10;

df\_array = zeros(n,1);

order\_array = zeros(n-2,1);

for i = 1:n

df\_array(i) = df(x0, h);

h = h/2;

end

for i = 1:n-2

order\_array(i) = log2((df\_array(i)-df\_array(i+1))/(df\_array(i+1)-df\_array(i+2)));

end

disp("order of numerical scheme:")

disp(order\_array)

**Console display:**

derivative: -0.12045

order of numerical scheme:

2.000011018672592

2.000003167886757

1.999996899588639

2.000031336607194

1.999758626038902

2.000290539479766

1.999433147986543

1.995473184108702

**Q4**

**Part(1) – Forward Euler method**

f = @(t,y) (t\*y-y^2/2)/t^2;

a = 1;

b = 3;

h = 1/128;

t\_array = linspace(a,b,(b-a)/h+1);

y\_array = zeros(1,length(t\_array));

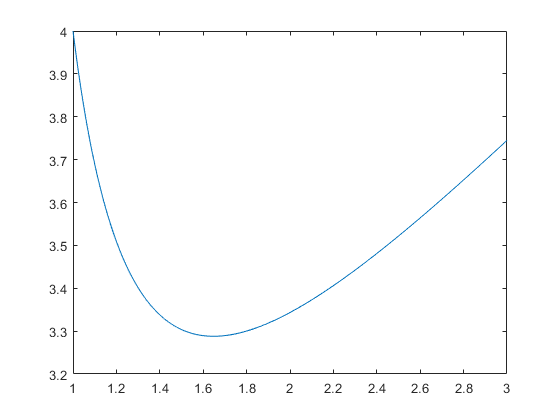
y\_array(1) = 4;

for i = 1:(b-a)/h

y\_array(i+1) = y\_array(i) + h\*f(t\_array(i), y\_array(i));

end

plot(t\_array, y\_array)



**Part(2) – Backward Euler method**

First you need to derive the actual scheme:

Given in the initial point , we should expect that is still very close to , thus when is taken arbitrarily small. Therefore, +ve sign should be taken.

f = @(t,y,h) (-(t^2-h\*t)+sqrt((t^2-h\*t)^2+2\*h\*t^2\*y))/h;

a = 1;

b = 3;

h = 1/128;

t\_array = linspace(a,b,(b-a)/h+1);

y\_array = zeros(1,length(t\_array));

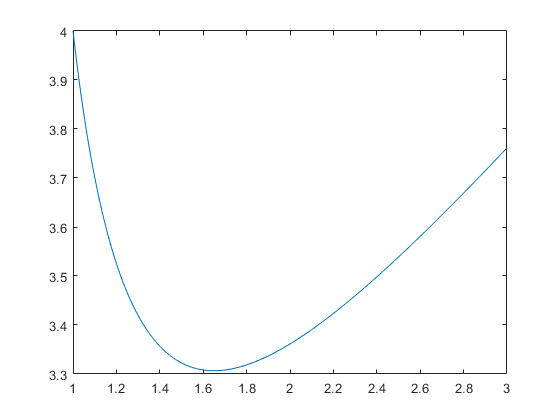
y\_array(1) = 4;

for i = 1:(b-a)/h

y\_array(i+1) = f(t\_array(i+1), y\_array(i), h);

end

plot(t\_array, y\_array)



**Part(3) - 4-th order Runge-Kutta method**

f = @(t,y) (t\*y-y^2/2)/t^2;

a = 1;

b = 3;

h = 1/128;

t\_array = linspace(a,b,(b-a)/h+1);

y\_array = zeros(1,length(t\_array));

y\_array(1) = 4;

for i = 1:(b-a)/h

z1 = f(t\_array(i), y\_array(i));

z2 = f(t\_array(i)+h/2, y\_array(i)+h\*z1/2);

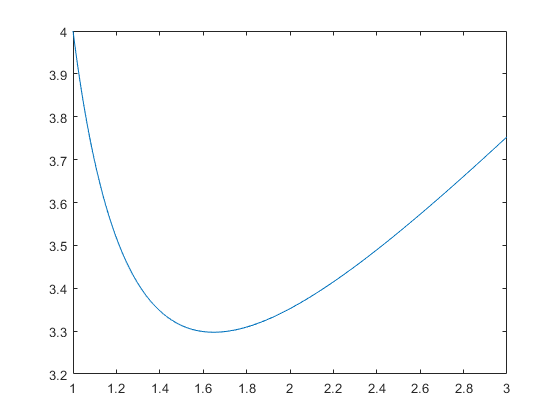
z3 = f(t\_array(i)+h/2, y\_array(i)+h\*z2/2);

z4 = f(t\_array(i)+h, y\_array(i)+h\*z3);

y\_array(i+1) = y\_array(i) + h/6\*(z1+2\*z2+2\*z3+z4);

end

plot(t\_array, y\_array)



**Q5**

Taylor expansion to time part

Therefore

Taylor expansion to position part

Therefore

So truncation error is