//Kamal Giri

//Scientific Computing

//MATH/COSC 3340

//Fall 2023

//Final Project

//Last Modified: 05/12/2023

Front Page:

A picture containing text, letter, font, paper

Description automatically generated

A picture containing text, screenshot, font, letter

Description automatically generated

Solutions:

#1:

Filename: steepestAscent.m

% Define the matrix A and the right-hand side b

A = [5 7 6 5; 7 10 8 7; 6 8 10 9; 5 7 9 10];

b = [-10; -14; -11; -8];

% A = [1 1 ; 2 1];

% b = [1;2];

% Define the initial guess

x0 = [0; 0; 0; 0];

% Define the maximum number of iterations and the tolerance

max\_iterations = 100000;

tolerance = 1e-7;

%Implementating the Algorithm

r0 = b-A\*x0;

fprintf(' k\t\tx\t\ty\t\tz\t\tw\t\tNorm\n');

for k = 1: max\_iterations

zk = A\*r0;

sk= (r0'\*r0)/(r0'\*zk);

xk = x0 + sk\*r0;

rk = r0 - sk\*zk;

if norm(rk) < tolerance

break;

end

if(k<=10)

fprintf(' %d\t %0.11f\t %0.11f\t %0.11f\t %0.11f\t %0.11f \n',k, xk(1),xk(2),xk(3),xk(4), norm(rk));

end

r0 = rk;

x0 = xk;

end

disp('Final Approximation:');

fprintf('%d\t %0.11f\t %0.11f\t %0.11f\t %0.11f\t %0.11f \n',k, xk(1),xk(2),xk(3),xk(4), norm(rk));

disp('Actual:');

xk = [1, -2, -1 ,1];

fprintf(' \t%0.11f\t %0.11f\t %0.11f\t %0.11f\t %0.11f \n', xk(1),xk(2),xk(3),xk(4), norm(rk));

Output:

A picture containing text, screenshot, font, receipt

Description automatically generated

#2:

Filename: sParabolicInter.m

clear all

format long

xplot = linspace(-2,-1,100);

f = @(x) abs(x.^2-2)+ abs(2\*x+3);

xL=-4;

xR=0;

m = (xL + xR)/2;

T= 10^-7;

r = 0.382;

%counting the iteration

i =0;

for n=1:100

nume = (f(xL) - f(m))\*(xR-m)^2 - (f(xR)-f(m))\*(m-xL)^2;

demue = (f(xL)- f(m))\*(xR-m) + (f(xR)- f(m))\*(m-xL);

xm = m + nume/(2\*demue);

if(f(xm)<f(m))

if(xm<m)

xR = m;

m = xm;

else

xL =m;

m = xm;

end

i = i+1;

if abs(xL- xR) <T

fprintf("Minimum of the function at x = %0.10f is %0.10f with %d iterations\n", xm, f(xm),i);

break;

end

else

%Using golden Search method

xm1 = xL + (xR-xL)\*r;

xm2 = xL + (xR-xL)\*(1-r);

if f(xm1)< f(xm2)

xR = xm2;

else

xL = xm1;

end

m = (xL+xR)/2;

i = i+1;

if abs(xL- xR) <T

fprintf("Minimum of the function at x = %0.10f is %0.10f with %d iterations\n", m, f(m),i);

break;

end

end

end

%Verifying the result with plot

plot(xplot, f(xplot), '-');

Output:

Minimum of the function at x = -1.4142136211 is 0.1715729239 with 61 iterations.

Graph:

A picture containing text, diagram, line, screenshot

Description automatically generated

#3.

Filename: backwardEulerM.m

%Backward Euler's Method

clear all;

format long;

% Define the differential equation and initial condition

f = @(u,t) 2 + sqrt(u) - 2\*t + 3;

u0 = 1;

% Define the time step size and time interval

dt = 0.05;

N = length(0:dt:2);

t = linspace(0,2,N);

u = zeros(size(t));

u(1) = u0;

% Apply the backward Euler's method with the Newton-Raphson method

for n = 1:N-1

tn = t(n);

un = u(n);

tn1 = t(n+1);

% Define the nonlinear equation to solve using Newton-Raphson method

nf = @(un1) un1 - un - dt\*f(un1,tn1);

dF = @(un1) 1 - dt/(2\*sqrt(un1-2\*tn1+3));

% Use the Newton-Raphson method to solve the nonlinear equation

un1 = un;

tol = 1e-6;

maxit = 100;

for i = 1:maxit

un1\_new = un1 - nf(un1)/dF(un1);

if abs(nf(un1\_new)) < tol

break;

end

un1 = un1\_new;

end

% Store the solution at the next time step

u(n+1) = un1;

end

% Plot the numerical solution and the exact solution

exact\_sol = 1 + 4\*t + t.^2/4;

plot(t, u, '-o', t, exact\_sol, '\*');

legend('Numerical Approximation', 'Exact solution');

xlabel('t')

ylabel('u(t)')

Output:

A picture containing text, line, plot, screenshot

Description automatically generated

I have also tried to do the Forward Euler’s Method just to verify my above implementation:

Filename: forwardEulerM.m

%Forward Euler Method

clear all;

format long;

% Define the differential equation and initial condition

f = @(u,t) 2 + sqrt(u) - 2\*t + 3;

u0 = 1;

% Define the time step size and time interval

dt = 0.05;

N = length(0:dt:2);

t = linspace(0,2,N);

u = zeros(size(t));

u(1) = u0;

% Using fordward Euler's Method

for n = 1:N-1

tn = t(n);

un = u(n);

un1 = un+ dt\*f(un, tn);

u(n+1) = un1;

end

% Plot the numerical solution and the exact solution

exact\_sol = 1 + 4\*t + t.^2/4;

plot(t, u, '-o', t, exact\_sol, '\*');

legend('Numerical Approximation', 'Exact solution');

xlabel('t')

ylabel('u(t)')

Output Graph:

A picture containing text, line, plot, screenshot

Description automatically generated