

//Kamal Giri
//Scientific Computing
//MATH/COSC 3340
//Fall 2023
//Final Project
//Last Modified: 05/12/2023

Front Page:

May 3, 2023
MATH 3340

Final Project

Due on or before Friday May 12th at 11:59PM


Submit online; make sure to sign the clause below

Sign by hand with blue or red pen then scan this page

All work with signed page on top should be one PDF file

Projects without signature as described above will receive no credit

Honesty Clause: I hereby testify that all the work on this project is solely my own. I have complied with the academic honesty guidelines as stated in the University regulations.

Your name Kamal Giri Signature  Date 05/12/2023

#1 Write a code of your own that solves the linear system of equations:

$$\begin{cases} 5x + 7y + 6z + 5w = -10 \\ 7x + 10y + 8z + 7w = -14 \\ 6x + 8y + 10z + 9w = -11 \\ 5x + 7y + 9z + 10w = -8 \end{cases}$$

using the steepest descent method and starting from the usual initial guess $\mathbf{x}^0 = [0, 0, 0, 0]^T$. For each iteration, print the iteration number k , the current approximation $\mathbf{x}^k = [x^k, y^k, z^k, w^k]^T$, and the norm of the residual at this current location. Start with $k = 0$ and produce a table with this information for the first ten iterations. Then run your code to obtain convergence, which should be declared when the norm of the residual is below 10^{-7} , and print the solution. Submit all pieces of code that you used to solve the problem, together with the aforementioned results.

#2 Use both successive parabolic interpolation (SPI) and the golden section search method to find the minimum of the function $f(x) = |x^2 - 2| + |2x + 3|$ on the interval $[-4, 0]$ with a tolerance of 10^{-8} and identical accuracy. You are again requested to do this with your own MATLAB codes which you must show, together with the output. Recall that the golden search method code was discussed in detail in class. Also, for parabolic interpolation, remember that some approximation steps may produce points that are not feasible and must be replaced. Therefore, write your SPI code such that it reverts to doing, for one step, a golden section search in that case; after a single corrective step, if such step is needed, the code should revert to parabolic interpolation.

#3 The implicit, backward Euler method for the general first order differential equation

$$\frac{du}{dt} = f(u, t)$$

is given by

$$u_{n+1} = u_n + (t_{n+1} - t_n) * f(u_{n+1}, t_{n+1}).$$

Use this method to solve numerically the differential equation

$$\frac{du}{dt} = 2 + \sqrt{u - 2t + 3}$$

subject to the initial condition $u(0) = 1$. Use a constant time step $\Delta t = t_{n+1} - t_n = 0.05$ and advance the solution from $t = 0$ to $t = 2$. Turn in the code and the plot of the numerical approximation to $u(t)$ versus t . On the plot, compare your numerical solution with the exact solution $u(t) = 1 + 4t + t^2/4$; use markers for the numerical solution.

NOTE: You will need to solve a nonlinear equation at each time step. Solutions that do not perform this task as required will not receive credit.

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];

```
max_iterations = 100000;  
tolerance = 1e-7;
```

%Implementating the Algorithm

```
r0 = b-A*x0;
fprintf(' k\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*zg;
    if norm(rk) < tolerance
        break;
    end
    if(k<=10)
        fprintf(' %d\t %0.11ft %0.11ft %0.11ft %0.11ft %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 %0.11ft %0.11ft %0.11ft %0.11ft %0.11f \n',k, xk(1),xk(2),xk(3),xk(4), norm(rk));

disp('Actual:');
xk = [1, -2, -1 ,1];
fprintf(' \t%0.11ft %0.11ft %0.11ft %0.11ft %0.11f \n', xk(1),xk(2),xk(3),xk(4), norm(rk));
```

Output:

```
>> steepestDescent
      k          x          y          z          w          Norm
      1      -0.34342424675  -0.48079394545  -0.37776667143  -0.27473939740  4.09859901768
      2      -0.59994804994  -0.85096499322  -0.14530900610   0.37408540034  4.49186835466
      3      -0.67043293964  -0.94577562223  -0.23181608660   0.32311945699  0.90250898499
      4      -0.72646632547  -1.00279006514  -0.23206315840   0.50709438333  1.19588927757
      5      -0.74420625363  -1.02373538824  -0.26319634201   0.49515848347  0.35894598021
      6      -0.74677566189  -1.00024564804  -0.30931102303   0.57804102797  0.81667382341
      7      -0.75766280777  -1.01270611933  -0.33214791543   0.56852881642  0.25909689397
      8      -0.75020275851  -0.98335927936  -0.37220237503   0.61771036371  0.61190876354
      9      -0.75825068090  -0.99288451320  -0.38930639674   0.61068498620  0.19426921747
     10      -0.75177364243  -0.97218265641  -0.41931031772   0.64824467404  0.45942872469
Final Approximation:
7107      0.99999227986  -1.99999533807  -0.99999806190   0.99999884956  0.00000009989
Actual:
      1.00000000000      -2.00000000000     -1.00000000000   1.00000000000  0.00000009989
```

#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
```

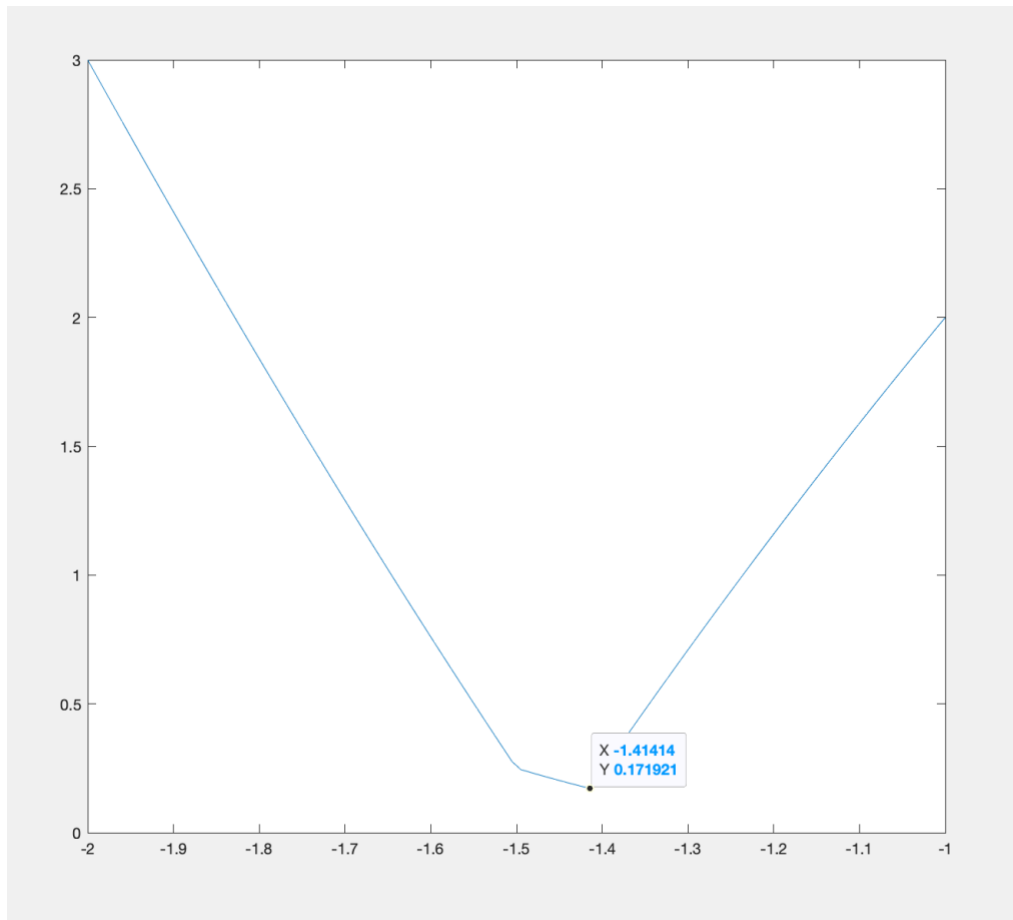
```
%Verifying the result with plot
```

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

Output:

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

Graph:



#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
```

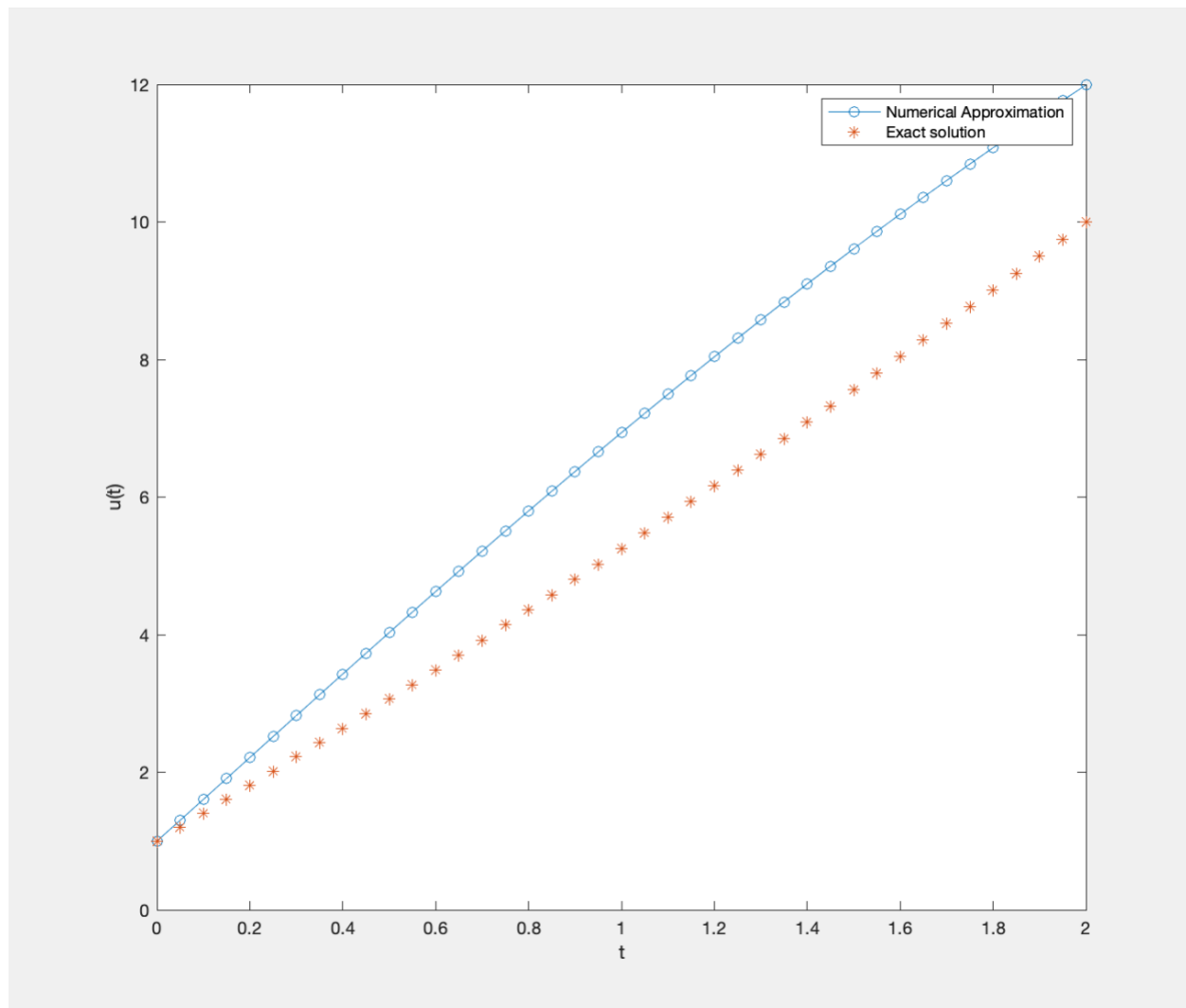
```
% Plot the numerical solution and the exact solution
```

```
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:



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 forward 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:

