Name, Surname: Cansu YANIK

Student Number: 150170704

ITU Computer and Informatics Faculty

BLG 202E Numerical Methods in CE 2018-2019 Spring

Homework-2

1.

```matlab
function roots = FindRoots( func, a, b, nprobe, tol )
    %Since I need the initial interval values, I save them.
    ai = a; bi = b;
    step = (b - a)/(nprobe - 1);
    x = (a : step : b);

    syms x
    %first derivative of the function
    dfunc = matlabFunction(diff(func(x)));

    xf = a;           %initial value for newton method
    roots=[];         %For saving the roots
    interval = [];    %For saving other interval which may have a root
    flag = 0;         %For checking other interval if a root is found

    flag2 = 0;        %Firstly I search the roots for the half of the function
                      %then I search them for other half. This variable is
                      %for the determine this situation.

    while(1)
        %Applies the newton method

        fxf = func(xf); dfxf = dfunc(xf);
        if (isnan(fxf))      %if the value is undefined
            fxf = 1;
        end

        xl = xf - round(fxf/dfxf);
        fxl = func(xl);
        if (isnan(fxl))       %if the value is undefined
            fxl = 1;
        end

        %Finding root criteria
        if and(abs(fxl) < tol, abs(xl-xf) < tol * (1+abs(xl)))
            if ~ismember(xl,roots)          %If the root is not already found,
                roots = [roots; xl];        %Saves the root to vector
            end
            flag = 1;
            if(isempty(interval))   %If there is not any interval left that we must chec
                flag2=flag2+1;      %I can look the other half of the function
                flag = 0;
                a = ai; b = bi;     %I need the initial interval values.
                step = (b - a)/(nprobe - 1);
                x = (a : step : b);
                syms x
                dfunc = matlabFunction(diff(func(x)));
                xf = a;
                if(flag2==2)        %If I check both two halfs, now I can stop checking.
                    break;
                end
            end
        else
            xf = xl;                %If a root is not found, continue to do steps
        end
        %If an iterate is reached for which there is no sufficient decrease
        if abs(func(xl)) >= 0.5*abs(func(xf))
            for i=1:3               %Does three bisection steps, finds midpoint
                p = (a+b)/2;
                fp = func(p); fa = func(a); fb = func(b);
                if (isnan(fp))  %if the value is undefined
                    fp = 1;
                end

                %Checks the signs to be able to understand existance of the root
                if flag2==0     %Finds the roots which are in the first half of the function
                    if fa*fp < 0
                        if fb*fp < 0
                            %To be able to check other interval, I saves it into a vector
                            interval = [a,b,p; interval];
                        end
                        b = p; fb = fp;
                    else
                        a = p; fa = fp;
                    end
                    %if the current interval gives a root, checks other interval
                    if flag == 1
                        a = interval(3); b = interval(2);
                        fa = fp;
                        flag = 0; interval(1,:)=[];
                    end
                else   %Now, Finds other roots which are in the second half of the function
                    if fb*fp < 0
                        if fa*fp < 0
                            %To be able to check other interval, I saves it into a vector
                            interval = [a,b,p; interval];
                        end
                        a = p; fa = fp;
                    else
                        b = p; fb = fp;
                    end
                    %if the current interval gives a root, checks other interval
                    if flag == 1
                        a = interval(1); b = interval(3);
                        fb = fp;
                        flag = 0; interval(1,:)=[];
                    end
                end
            end
            step = (b - a)/(nprobe - 1);    %Updates function with the new interval
            x = (a : step : b);
            syms x
            dfunc = matlabFunction(diff(func(x)));
            xf = a;
        end
    end
end
```

a. My function takes five inputs which are the function, two interval values, number of nprobe and tolerance tol. After implementing my program, I tried to verify my program by finding the two roots of the function, $f(x) = 2cosh(x/4) - x$, starting the search with $[a, b] = [0,10]$ and $nprobe = 10$. My program works well and gives two roots for this question.

```
>> roots = FindRoots(@(x) (2*cosh(x/4) - x), 0, 10, 10, 10 .^ (-7));
>> roots

roots =

    2.3576
    8.5072
```

b. For this function which is given below, my program gives six roots which are shown in the figure below.

$$f(x) = f(x) = \begin{cases} \dfrac{\sin(x)}{x}, & x \neq 0 \\ 1, & x = 0 \end{cases}$$

in the interval $[-10,10]$ for $tol = 10^{-7}$

```
>> roots = FindRoots(@(x) (sin(x)./x), -10, 10, 1000, 10 .^ (-7));
>> roots

roots =

   -3.1416
    6.2832
    9.4248
    3.1416
   -6.2832
   -9.4248
```

Name, Surname: Cansu YANIK

Student Number: 150170704

2.

Function	**Proper Method**	**Explanation**
a. $f(x) = x - 1$ on the interval [0,2.5].	Newton Method	Newton's method needs smoothness of the function. To be able to use this method, the function must have first and second derivatives, and also the function must be continuous.
b. $f(x)$ is given in Figure 1 on [0, 4].	Bisection Method	Bisection method needs the function to be merely continuous, and it does not use derivatives of the function. Just simple and minimal assumptions on the function are required like this function. It is good to use this method in functions which has not got derivatives.
c. $f(x) \in C^5[0.1,0.2]$, the derivatives of f are all bounded in magnitude by 1, and $f'(x)$ is hard to specify explicitly or evaluate.	Secant Method	Since for this function $f'(x)$ is hard to specify explicitly or evaluate, using Secant method has more advantage. It is possible to evaluate only the function itself.

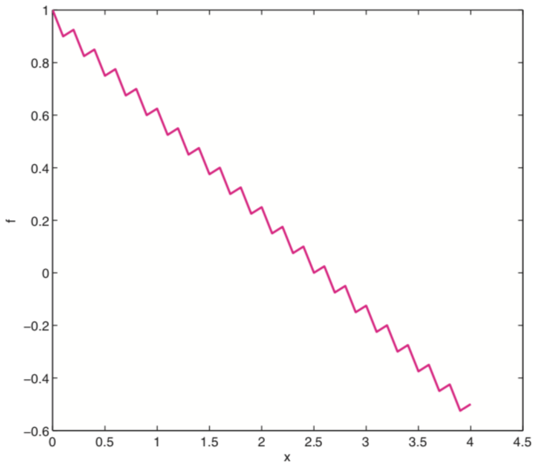

Figure 1: Graph of an anonymous function

Name, Surname: Cansu YANIK

Student Number: 150170704

3.

```
%
%  [ a(1)  v1(1)                                  ] [  X(1)  ]   [  b(1)  ]
%  [ v2(2)  a(2)  v1(2)                           ] [  X(2)  ]   [  b(2)  ]
%  [       v2(3)  a(3)  v1(3)                     ] [        ]   [        ]
%  [              ...   ...   ...                 ] [  ...   ] = [  ...   ]
%  [                    ...    ...    ...         ] [        ]   [        ]
%  [                        v2(n-1) a(n-1) v1(n-1) ] [ X(n-1) ]   [ b(n-1) ]
%  [                                v2(n)   a(n)   ] [  X(n)  ]   [  b(n)  ]
%First value of the v1 and the last value of the v2 must be 0. Each vector must
%be given as an input.
%

function X = Tridiagonalsys(v1,a,v2,b)

for index = 2:length(a)
    a(index) = a(index) - (v1(index) / a(index-1)) * v2(index-1);
    b(index) = b(index) - (v1(index) / a(index-1)) * b(index-1);
end

X(length(a)) = b(length(a)) / a(length(a));

for index = length(a)-1 : -1 : 1
    X(index) =  (b(index) - v2(index) * X(index+1)) / a(index);
end

```

The cost of this program is $O(n)$. Program takes four vector as inputs. v1, a and v2 vectors form the tridiagonal matrix A, b is right-hand-side vector. This program calculates and return $\boldsymbol{x} = A^{-1}\boldsymbol{b}$ using a Gaussian elimination variant. First for loop does the forward elimination, after that by applying back substitution, x vector can be found.

After implementation of the code, I tried it on the matrix defined by $n = 10$, $a_{i-1,i} = a_{i+1,i} = -i$, and $a_{i,i} = 3i$ for all i such that the relevant indices fall in the range 1 to n. I chose a right-hand-side vector like $\boldsymbol{b} = 1$. The matrixes and the result is given below:

```
>> v1=[0;-1;-2;-3;-4;-5;-6;-7;-8;-9];
>> v2=[-1;-2;-3;-4;-5;-6;-7;-8;-9;0];
>> a=[3;6;9;12;15;18;21;24;27;30];
>> b=[1;1;1;1;1;1;1;1;1;1];
>> x = Tridiagonalsys(v1,a,v2,b);
>> x

x =

    0.4412    0.3237    0.2504    0.2021    0.1685    0.1439    0.1246    0.1077    0.0889    0.0600

>> |
```

Name, Surname: Cansu YANIK

Student Number: 150170704

4.

$$A = \begin{pmatrix} 5 & 6 & 7 & 8 \\ 0 & 4 & 3 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & -2 \end{pmatrix}$$

a. The matrix A can be decomposed using partial pivoting as $PA = LU$, where U is upper triangular, L is unit lower triangular, and P is a permutation matrix. Firstly, I will try to find Upper (U) triangular matrix (Thus I can solve the problem by using backward substitution). Therefore, I will use pivoting strategies. I need to modify the algorithm of Gaussian elimination by pivoting.

$$P^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},\ M^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},\ A^{(1)} = M^{(1)} P^{(1)} A = \begin{pmatrix} 5 & 6 & 7 & 8 \\ 0 & 4 & 3 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & -2 \end{pmatrix}$$

$$(l_{21} = 0, l_{31} = 0, l_{41} = 0)$$

$$P^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},\ M^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},\ A^{(2)} = M^{(2)} P^{(2)} A^{(1)} = \begin{pmatrix} 5 & 6 & 7 & 8 \\ 0 & 4 & 3 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & -2 \end{pmatrix}$$

$$(l_{32} = 0, l_{42} = 0)$$

$$P^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix},\ M^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},\ A^{(3)} = M^{(3)} P^{(3)} A^{(2)} = \begin{pmatrix} 5 & 6 & 7 & 8 \\ 0 & 4 & 3 & 2 \\ 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$(l_{43} = 0)$$

$$U = MP = A^{(3)} \Rightarrow U = \begin{pmatrix} 5 & 6 & 7 & 8 \\ 0 & 4 & 3 & 2 \\ 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & 1 \end{pmatrix},\ P = P^{(1)} P^{(2)} P^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$PA = LU \Rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 5 & 6 & 7 & 8 \\ 0 & 4 & 3 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & -2 \end{pmatrix} = \mathbf{L} \begin{pmatrix} 5 & 6 & 7 & 8 \\ 0 & 4 & 3 & 2 \\ 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow \mathrm{L} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

b. Backward substitution can be used to solve the problem.

$$\mathrm{LUx = Pb} \Rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 5 & 6 & 7 & 8 \\ 0 & 4 & 3 & 2 \\ 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x1 \\ x2 \\ x3 \\ x4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 26 \\ 9 \\ 1 \\ -3 \end{pmatrix}$$

$X_4 = 1$
$-x_3 - 2x_4 = -3 \Rightarrow x_3 = 1$
$4x_2 + 3x_3 + 2x_4 = 9 \Rightarrow x_2 = 1$
$5x_1 + 6x_2 + 7x_3 + 8x_4 = 26 \Rightarrow x_1 = 1$

$$\mathrm{X} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$