University of Waterloo

ECE204 Lab Report

Simulation Assignment #3

Section: 202

Kehan Xing (k5xing)

Jing Liu (j762liu)

Submission Date: 2019-10-31

Q1. Create a generic function to solve a system of nonlinear equation using newton Raphson method. The function should accept any acceptable initial condition, variables as symbolic variable and nonlinear equation. Use your program to solve the following two non-linear equations:

f=[4*x^2+y^2-13, x^2+y^2-10];

f=[2*x-y-exp(-x), -x+2*y-exp(-y)];

The iteration should stop when the relative approximate error is less than 0.5%.

Code:

```
%define symbolic variables
syms x;
syms y;
syms var;
var=[x;y];
%define symbolic function
syms f(x,y);
%system of functions to be evaluted
f=[2*x-y-exp(-x); -x+2*y-exp(-y)];
%initial value
init =[0.1;0.1];
%invoke the Newton Raphson function
newton(f,var,init);
function newton (f, var,init)
    %set up prev
    prev=init;
    %set up intial values for error
    error=ones(length(init),1);
    %looping until satisfy error tolerance
    while max(error)>0.5

        %get jacobian
        jac = jacobian(f, var);
        %evaluate jacobian matrix
        j_value=double(subs(jac,var,prev));
        %evaluate f
        f_value=double(subs(f,var,prev));
        %update result
        result = prev - j_value\f_value;
        %update error
        error=100*abs((result-prev)./result);
        %update prev
        prev=result;
    end
     disp(result);

end
```

Display:

For f=[2*x-y-exp(-x), -x+2*y-exp(-y)]

Editor - N:\2A\ECE204\Lab3\q1.m

q1.m | q2.m | +

```matlab
%define symbolic variables
syms x;
syms y;
syms var;
var=[x;y];
%define symbolic function
syms f(x,y);
%system of functions to be evaluted
f=[2*x-y-exp(-x); -x+2*y-exp(-y)];
%initial value
init =[0.1;0.1];
%invoke the Newton Raphson function
newton(f,var,init);
function newton (f, var,init)
    %set up prev
    prev=init;
    %set up intial values for error
    error=ones(length(init),1);
    %looping until satisfy error tolerance
    while max(error)>0.5

        %get jacobian
        jac = jacobian(f, var);
        %evaluate jacobian matrix
        j_value=double(subs(jac,var,prev));
        %evaluate f
        f_value=double(subs(f,var,prev));
        %update result
        result = prev - j_value\f_value;
        %update error
        error=100*abs((result-prev)./result);
        %update prev
        prev=result;
    end
     disp(result);

end
```

Command Window

```
>> q1
    0.5671
    0.5671

>>
```

For f=[4*x^2+y^2-13, x^2+y^2-10];

Editor - N:\2A\ECE204\Lab3\q1.m

q1.m q2.m

```matlab
%define symbolic variables
syms x;
syms y;
syms var;
var=[x;y];
%define symbolic function
syms f(x,y);
%system of functions to be evaluted
%f=[2*x-y-exp(-x); -x+2*y-exp(-y)];
f=[4*x^2+y^2-13; x^2+y^2-10];
%initial value
init =[0.1;0.1];
%invoke the Newton Raphson function
newton(f,var,init);
function newton (f, var,init)
    %set up prev
    prev=init;
    %set up intial values for error
    error=ones(length(init),1);
    %looping until satisfy error tolerance
    while max(error)>0.5

        %get jacobian
        jac = jacobian(f, var);
        %evaluate jacobian matrix
        j_value=double(subs(jac,var,prev));
        %evaluate f
        f_value=double(subs(f,var,prev));
        %update result
        result = prev - j_value\f_value;
        %update error
        error=100*abs((result-prev)./result);
        %update prev
        prev=result;
    end
     disp(result);

end
```

Command Window

```
>> q1
    1.0000
    3.0000

>>
```

Q2. Using the information in example 9, in the set of notes titles “Roots of nonlinear equations”, find the temperature of an RTD that measure a resistance of:

75Ω and 250Ω

Use both bisection and Newton Raphson method. The iteration should stop when the relative approximate error is less than or equal 0.1%. Show the needed number of iterations of both methods. The program should be general and accept any resistance value.

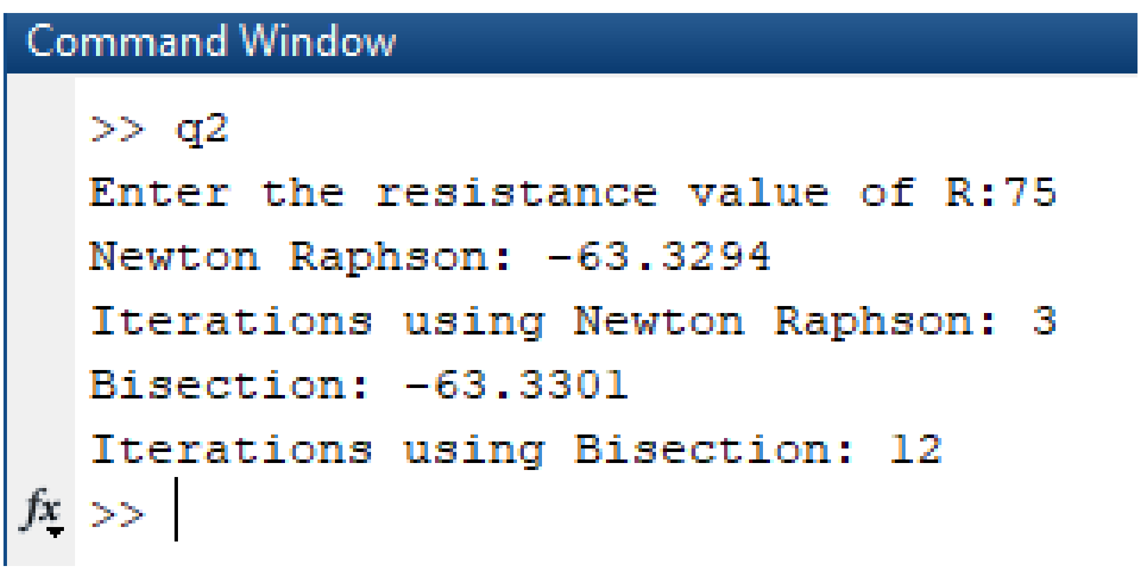
Code:

```
syms T;
syms f1(T);
syms f2(T);
%take resistance
R = input('Enter the resistance value of R:');

f1= 5.775*(10^(-7))*T^2-3.9083*(10^(-3))*T+(R/100-1);
f2=5.775*(10^(-7))*T^2-3.9083*(10^(-3))*T+4.183*(10^(-12))*(T-100)*T^3+R/100-
1;

%check which function to use depends on R
if R<100
    newrap(f2, -100,1);
    bisection(f2,-200,0,1);
else
     newrap(f1, 300,1);
     bisection(f1,0,850,1);
end

%bisection function
function bisection(f,init_1,init_2,counter)
    value_1=subs(f,init_1);
    value_2=subs(f,init_2);

    %update prev
    prev=init_1;

    %new root
    result=(init_1+init_2)/2;

    %get f(T) using new root
    value_new=subs(f,result);

    %recursively call biesection function until relative error < 0.1%
    if(abs((result-prev)/result)<=0.001)
        disp(['Bisection: ',num2str(double(result))]);
         disp(['Iterations using Bisection: ', num2str(counter)]);
    %update the range of the root
    elseif value_new*value_1<0
```

```
        %keep tracking of iterations
        counter = counter + 1;
        bisection(f,init_1,result,counter);
    elseif value_new*value_2<0
        counter = counter + 1;
        bisection(f,result,init_2,counter);
    end
end

%Newton Raphson function
function newrap(f, init, counter)
    %update prev
    prev = init;
    %differentiate the function
    df = diff(f);
    %new root
    new_T = double(prev - subs(f, prev)/subs(df, prev));

    %recursively call Newton Raphson function until relative error < 0.1%
    if(abs(new_T - prev) <= 0.001)
        disp(['Newton Raphson: ' ,num2str(double(new_T))]);
        disp(['Iterations using Newton Raphson: ', num2str(counter)]);
    else
        %keep tracking of iterations
        counter = counter + 1;
        newrap(f, new_T,counter);
    end


end
```

Display:

For R = 75Ω



For R = 250Ω

Command Window

```
>> q2
Enter the resistance value of R:250
Newton Raphson: 408.45
Iterations using Newton Raphson: 3
Bisection: 408.606
Iterations using Bisection: 12
>>
```