ECE 204 Simulation Assignment 2

Phoebe Luo
University of Waterloo

Results and Discussion

Q1. Using the information in example 9, in the set of notes titled “Roots of nonlinear equations”, find the temperature of an RTD that measure a resistance of: 55Ω and 260Ω.

- The program should be general and accept any resistance value.
- The code should ask the user to input the resistance value.
- Use **both bisection and Newton Raphson** method.
- The iterations should stop when the absolute relative approximate error percentage is less than or equal **0.05%**.

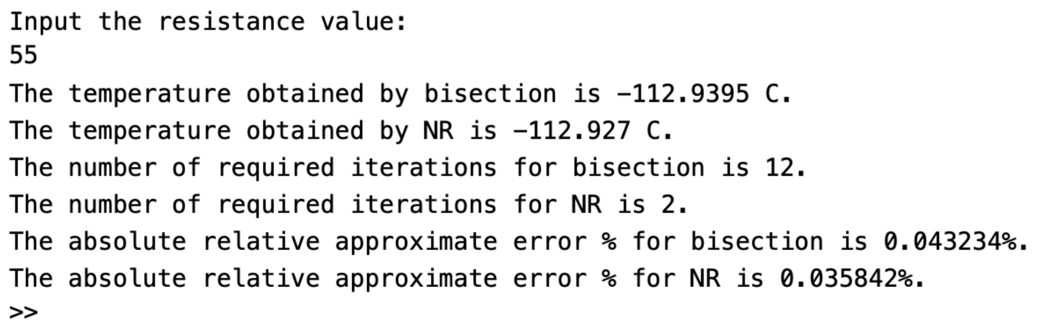
- The code is:

```
format shortg;
clc;

% Ask User to Input R value
prompt = 'Input the resistance value: ';
R = input(prompt);

% Bisection Method
Tm_l = 0; % keep track of the last Tm for error calculation
error_1 = 1; % absolute relative approximate error percentage
count_1 = 0; % keep track of the number of iterations

if R > 100
min = 0; % the left range for the iteration if inputed R is higher than 100
max = 850; % the right range for the iteration if inputed R is higher than
100
fmin = 100 * (1 + 3.9083 * 10^-3 * min - 5.775 * 10^-7 * min^2) - R; % f of
left range
fmax = 100 * (1 + 3.9083 * 10^-3 * max - 5.775 * 10^-7 * max^2) - R; % f of
left range
while error_1 > 0.05
Tm = (min + max)/2;  % calculated temperature Tm for the iteration
fTm = 100 * (1 + 3.9083 * 10^-3 * Tm - 5.775 * 10^-7 * Tm^2) - R;
if fmin * fTm < 0 % if fmin * fTm < 0, change range to be [min, Tm]
max = Tm;
else % else, change range to be [Tm, max]
            min = Tm;
        end
        error_1 = abs((Tm - Tm_l) / Tm) * 100;
        Tm_l = Tm; % update Tm_l to keep track of Tm
        count_1 = count_1 + 1;
    end
else
    min = -200; % the left range for the iteration if inputed R is lower than
100
    max = 0; % the right range for the iteration if inputed R is lower than
100
```

```
    fmin = 100 * (1 + 3.9083 * 10^-3 * min - 5.775 * 10^-7 * min^2 - 4.183 *
10^-12 * (min - 100) * min^3) - R; % f of left range
    fmax = 100 * (1 + 3.9083 * 10^-3 * max - 5.775 * 10^-7 * max^2 - 4.183 *
10^-12 * (max - 100) * max^3) - R; % f of left range
    while error_1 > 0.05
        Tm = (min + max)/2;  % calculated temperature Tm for the iteration
        fTm = 100 * (1 + 3.9083 * 10^-3 * Tm - 5.775 * 10^-7 * Tm^2 - 4.183 *
10^-12 * (Tm - 100) * Tm^3) - R;
        if fmin * fTm < 0 % if fmin * fTm < 0, change range to be [min, Tm]
            max = Tm;
        else % else, change range to be [Tm, max]
            min = Tm;
        end
        error_1 = abs((Tm - Tm_l) / Tm) * 100;
        Tm_l = Tm; % update Tm_l to keep track of Tm
        count_1 = count_1 + 1;
    end
end

% Newton Raphson Method
error_2 = 1; % absolute relative approximate error percentage
count_2 = 1; % keep track of the number of iterations (starts at 1 for the
1st iteration outside of the while loop)

if R > 100
    T_i = 450; % the initial T of range [0, 850)
    T = T_i - (5.775 * 10^-7 * T_i^2 - 3.9083 * 10^-3 * T_i + (R/100) - 1) /
(2 * 5.775 * 10^-7 * T_i - 3.9083 * 10^-3);
    % get T of first iteration using T = T_i - f(T_i)/f'(T_i)
    T_l = T; % update T_l to keep track of T
    while error_2 > 0.05
        T = T_l - (5.775 * 10^-7 * T_l^2 - 3.9083 * 10^-3 * T_l + (R/100) -
1)/(2 * 5.775 * 10^-7 * T_l - 3.9083 * 10^-3);
        % get T using T = T_l - f(T_l)/f'(T_l)
        error_2 = abs((T - T_l) / T) * 100;
        T_l = T; % update T_l to keep track of T
        count_2 = count_2 + 1;
    end
else
    T_i = -100; % the initial T of range (-200, 0)
    T = T_i - (4.183 * 10^-12 * T_i^4 - 4.183 * 10^-10 * T_i^3 + 5.775 * 10^-
7 * T_i^2 - 3.9083 * 10^-3 * T_i + R/100 -1) / (4 * 4.183 * 10^-12 * T_i^3 -
3 * 4.183 * 10^-10 * T_i^2 + 2 * 5.775 * 10^-7 * T_i - 3.9083 * 10^-3);
    % get T of first iteration using T = T_i - f(T_i)/f'(T_i)
    T_l = T; % update T_l to keep track of T
    while error_2 > 0.05
        T = T_l - (4.183 * 10^-12 * T_l^4 - 4.183 * 10^-10 * T_l^3 + 5.775 *
10^-7 * T_l^2 - 3.9083 * 10^-3 * T_l + R/100 -1) / (4 * 4.183 * 10^-12 *
T_l^3 - 3 * 4.183 * 10^-10 * T_l^2 + 2 * 5.775 * 10^-7 * T_l - 3.9083 * 10^-
3);
        % get T using T = T_l - f(T_l)/f'(T_l)
        error_2 = abs((T - T_l) / T) * 100;
        T_l = T; % update T_l to keep track of T
        count_2 = count_2 + 1;
    end
end

DISP1 = ['The temperature obtained by bisection is ', num2str(Tm), ' C.'];
DISP2 = ['The temperature obtained by NR is ', num2str(T), ' C.'];
DISP3 = ['The number of required iterations for bisection is ',
num2str(count_1), '.'];
DISP4 = ['The number of required iterations for NR is ', num2str(count_2),
'.'];
DISP5 = ['The absolute relative approximate error % for bisection is ',
num2str(error_1), '%.'];
DISP6 = ['The absolute relative approximate error % for NR is ',
num2str(error_2), '%.'];

disp(DISP1);
disp(DISP2);
disp(DISP3);
disp(DISP4);
disp(DISP5);
disp(DISP6);
```

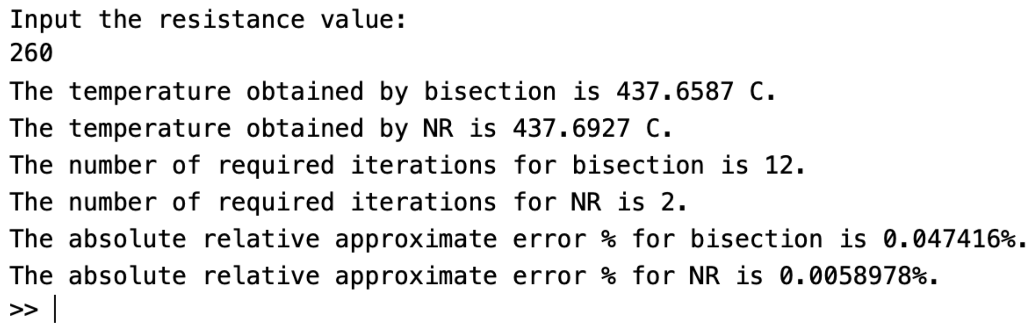
- The output is:

R = 55Ω



```
Input the resistance value:
55
The temperature obtained by bisection is -112.9395 C.
The temperature obtained by NR is -112.927 C.
The number of required iterations for bisection is 12.
The number of required iterations for NR is 2.
The absolute relative approximate error % for bisection is 0.043234%.
The absolute relative approximate error % for NR is 0.035842%.
>>
```

R = 260Ω



```
Input the resistance value:
260
The temperature obtained by bisection is 437.6587 C.
The temperature obtained by NR is 437.6927 C.
The number of required iterations for bisection is 12.
The number of required iterations for NR is 2.
The absolute relative approximate error % for bisection is 0.047416%.
The absolute relative approximate error % for NR is 0.0058978%.
>> |
```

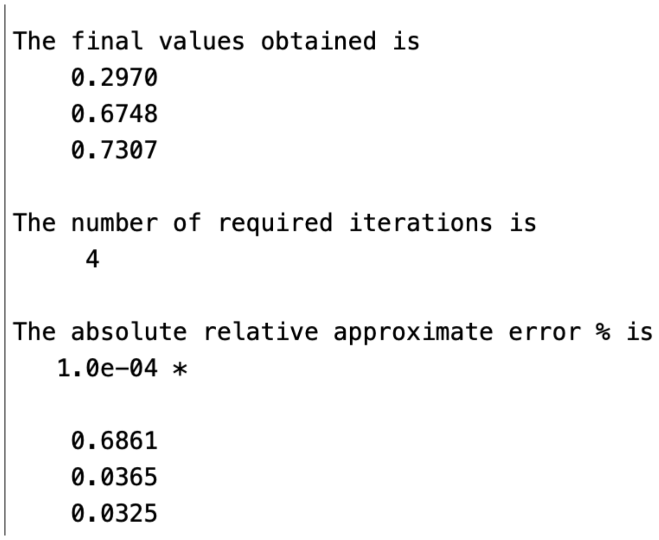
Q2. Create a code to solve a system of nonlinear equation using Newton Raphson method. The code should accept any acceptable initial condition, variables as symbolic variable and nonlinear equations. Use your program to solve the following non-linear equations from "Practice Questions 4 – Question number 3:"

$$x^3 - 10x + y - z + 3 = 0$$
$$y^3 + 10y - 2x - 2z - 5 = 0$$
$$x + y - 10z + 2\sin(z) + 5 = 0$$

- The code is:

```
clc;

% Symbolic Expressions
syms x y z
f1 = x^3 - 10*x + y - z + 3;
f2 = y^3 + 10*y - 2*x - 2*z - 5;
f3 = x + y - 10*z + 2*sin(z) +5;

x_c = [0, 0, 0]; % the current x in the iteration
x_l = [1, 1, 1]; % keep track of last x with a initial value of [x, y, z]
count = 0; % keep track of the number of iterations initialized as 1
error = [1, 1, 1]; % absolute relative approximate error percentage
initialized as 1 for each row

% The Jacobian matrix
Fx = [f1; f2; f3];
J = jacobian(Fx);

while (error(1,1) >= 0.1) || (error(1,2) >= 0.1) || (error(1,3) >= 0.1)
% get Fx output for the iteration
a = double(subs(f1, [x, y, z], x_l));
b = double(subs(f2, [x, y, z], x_l));
c = double(subs(f3, [x, y, z], x_l));
Fx_o = [a; b; c];
% get J output for the iteration
d = double(subs(J(1,:), [x, y, z], x_l));
e = double(subs(J(2,:), [x, y, z], x_l));
f = double(subs(J(3,:), [x, y, z], x_l));
J_o = [d; e; f];
% Perform Gauss Elimination to get delta x
    % forward elimination
    A = [J_o Fx_o]; % combine J_o, Fx_o together to do operation
    row_num = 3; % the number of rows
    row_swap = 1; % keep track of the row number that will be swapped with
the cur_row
    ope_num = row_num - 1; % the number of operations in total
    cur_row = 2; % keep track of the current row of subtraction for this
operation (initially at row 2)
    cur_col = 1; % keep track of the current column of subtraction (initially
at column 1)
    count_t = cur_row; % keep track of the current row being manipulated
    for a = 1:ope_num
        count_t = cur_row;
        while count_t < (row_num+1)
            A(count_t,:) = A(count_t,:)-
A(a,:).*(A(count_t,cur_col)/A(a,cur_col));
            count_t = count_t + 1;
        end
        % row swap
        [~,row_swap] = max(abs(A(cur_row:row_num,cur_col+1))); % find the row
number with the max
        temp = A(cur_row,:);
        A(cur_row,:) = A(row_swap+cur_row-1, :);
        A(row_swap+cur_row-1,:) = temp;
        cur_row = cur_row + 1;
        cur_col = cur_col + 1;
    end

    % back substitution
    delta_x = zeros (row_num,1); % solution matrix of delta x
    x_s = row_num; % x for solving from the bottom right diagonally
    num_sub = 0; % number of subtraction required for A(x, row_sum)
    cur_sub = num_sub; % keep track of needed subtraction left for A(x,
row_sum)

    while x_s > 0
        while cur_sub > 0
            n = row_num - (cur_sub - 1);
            A(x_s,row_num+1) = A(x_s, row_num+1) -  delta_x(n, 1)*A(x_s, n);
            cur_sub = cur_sub - 1;
        end
        delta_x(x_s,1) = A(x_s,row_num+1)/A(x_s,x_s);
        num_sub = num_sub + 1;
        cur_sub = num_sub;
        x_s = x_s - 1;
    end
    % convert delta_x to its transpose for subs format
    delta_x = transpose(delta_x);
    x_c = x_l - delta_x;
    error(1,1) = abs((x_c(1,1) - x_l(1,1)) / x_c(1,1)) * 100;
    error(1,2) = abs((x_c(1,2) - x_l(1,2)) / x_c(1,2)) * 100;
    error(1,3) = abs((x_c(1,3) - x_l(1,3)) / x_c(1,3)) * 100;
    x_l = x_c;
    count = count + 1;
end

% convert x_c, error to their transpose for display format
x_c = transpose(x_c);
error = transpose(error);

disp("The final values obtained is");
disp(x_c);
disp("The number of required iterations is");
disp(count);
disp("The absolute relative approximate error % is");
disp(error);
```

- The output is:



```
The final values obtained is
    0.2970
    0.6748
    0.7307

The number of required iterations is
     4

The absolute relative approximate error % is
   1.0e-04 *

    0.6861
    0.0365
    0.0325
```