GNG1106 – Fundamentals of Engineering Computation
Course Project

Project Name

Design of an RLC Electrical Circuit

Date: 17/11/2016

1 Problem Identification and Statement

Many problems in engineering require solving complicated equations. In some cases, the equation may not be solvable by simple algebraic methods. Indeed, we may need to refer to **numerical methods**. For example, equations which contain variables that aren't easy to isolate may need the use of numerical methods. For example, we can create a function out of an equation and attempt to find its roots (root finding methods). To do that, we can go over a range of numbers and test for which values the equation works, and thus finds solutions. We can also for example repeatedly divide an interval in half and select a subinterval in which a root lies.

In our case, the design problem is an electrical engineering one, and involves determining the proper resistor R to dissipate energy at a specified rate, for selected values for inductance L and capacitance C and a percentage of original charge p_c. It is possible to derive an equation which is a function of a variable resistance, and which contains the (constant) parameters which we have previously stated. Specific solutions to a complicated equation as such are very hard to find using algebraic methods. In fact we will need numerical methods for our problem solving.

We will state the equations and laws involved in the problem in section 2 of this deliverable, and develop a relatively complex code later on which will solve our problem and find the roots of the equation in question.

What are the physical laws at play in the case of our problem? To what mathematical equations do they lead? What relevant information is required to solve the problem? What numerical method shall we use to do so, and what kind of data do we need? Finally, what is the input information to the problem and the expected output?

Section 2 shall give a very general answer to each of these questions, and we will get more and more detailed at every step of this project. We will finally be able to develop a code to solve the problem, giving us an output for every input that a user wishes to enter.

2 Gathering of Information and Input/Output Description

2.1 SubSection 1

Using **Kirchhoff's second law**, which states that the total sum of voltage drops around a closed electrical circuit is zero, it is possible to derive an equation that describes the time variation of the charge on a capacitor in an RLC electrical circuit:

$$q(t) = q_0 e^{-Rt/(2L)} \cos\left[\left(\sqrt{\frac{1}{LC} - \left(\frac{R}{2L}\right)^2}\right) t\right] \quad \text{(Equation 1)}$$

The rate of dissipation would be defined as the time, t_d, it takes for the charge on the capacitor to reach a desired percentage of its original charge, $p_c = q/q_0$. That is, $q(t_d) = p_c q_0$. By substitution, we obtain:

$$q(t_d) = p_c q_0 = q_0 e^{-Rt_d/(2L)} \cos\left[\left(\sqrt{\frac{1}{LC} - \left(\frac{R}{2L}\right)^2}\right) t_d\right]$$

$$p_c = e^{-Rt/(2L)} \cos\left[\left(\sqrt{\frac{1}{LC} - \left(\frac{R}{2L}\right)^2}\right) t_d\right] \quad \text{(Equation 2)}$$

2.2 SubSection 2

As mentioned, we will use numerical methods to solve for R in the previous equation. Numerical methods can provide very good results by using a root finding method, that is finding the root of the function g(R), which is a simple rearrangement of Equation (2):

$$g(R) = e^{-Rt_d/(2L)} \cos\left[\left(\sqrt{\frac{1}{LC} - \left(\frac{R}{2L}\right)^2}\right) t_d\right] - p_c \quad \text{(Equation 3)}$$

This will give the value of R which satisfies Equation (2). In this case, we need to simply find the roots of g(R), otherwise stated as g(R) = 0. **This will give us the proper value of resistance R to dissipate energy knowing the rate of dissipation *t_d*, the inductance L, the percentage of original charge *p_c*, and the capacitance C.**

In order to do this, we will use, more specifically, the Bisection Method for Finding Roots. Using this method, we can repeatedly bisect an interval in half and select a subinterval in which a root lies. If a function changes sign over an interval, the function value at the midpoint is evaluated. The location of the root is then determined as lying at the midpoint of the subinterval within which the sign change occurs. The process is repeated to obtain refined estimates.

The advantages of this method is that it's a quite precise and accurate method of computing roots, and it involves a relatively simple algorithm.

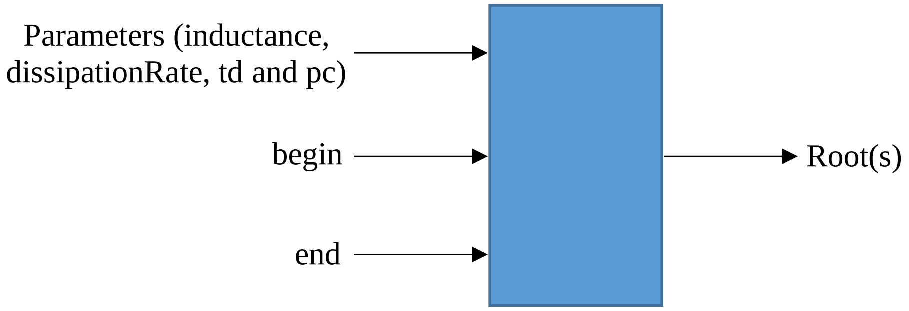
However, one drawback to this method is that it finds only one root at a time. It is therefore quite costly computationally costly if there are many roots involved. Another drawback is the fact that this method works well for functions which are not complex, and hyperbolae or parabolas. Even though it's accurate, this method remains a numerical method and is only an approximation of the actual solution to the equation.

2.3 SubSection 3

1. At the start of the program, the user will be asked to input a set of specific (constant) parameters in the equation used to compute the resistance R, which we will define in the input/output subsection right after. When new input values are given, the user is given the option to save them into a file; up to five sets of values can be stored, and each new inputted value will replace a value which was stored beforehand inside of the file. We will create the file beforehand in parallel to the development of the code.
2. The software first calculates an upper bound for the resistor using the square root term in the function g(R) (equation 3) and will then plot the function for root finding.
3. After this, the user will be allowed to select an appropriate interval for searching the root of the function. The user will be allowed to repeatedly provide upper and lower bounds (a different interval) to replot g(R) until he is satisfied with the bounds.
4. The program then uses a root-searching method, the bisection method which we have mentioned in order to get as close as possible to the root, which will be equal to the resistance. Multiple roots/solutions may exist, in which case the user will be asked if he wishes for the program to calculate the next root, by giving the user the choice to redefine the interval.

2.4 Input and Ouput

- Input:
  - Parameters: inductance L, capacitance C, dissipation rate t_d and percentage of charge p_c.
  - Beginning and end of the interval to search: begin, end.
- Output:
  - The real roots found in the search interval.


Parameters (inductance, dissipationRate, td and pc)
begin
end
Root(s)

3 Test Cases and Design

3.1 Test Cases

Input	Output	Description
Capacitance = 0.3 Inductance = 4 Dissipation rate = 0.5 Percentage of charge = 0.7 Start value = 0 End value = 6	R=2.915039	This is an example of a working test case which uses **in-between values of input**.
Inductance = 0.5 H Capacitance = 0.112 F Dissipation rate = 0.9 s Percentage of charge = 0.4 Start value = -4 End value = -2	R = -3.85475	This is an example of a function which contains a negative root. Since the resistance cannot be negative, the program will return an error message. **This tests invalid input values.**
Inductance = 2.5 Capacitance = 10 Dissipation rate = 0.1 Percentage of charge = 0.9 Start value = 4 End value = 6	The function is not defined on the given interval. Error will be given by the program.	This is an example of a test case where the function is not defined within the boundaries given by the user. **Tests invalid values.**
Inductance = 100 Capacitance = 300 Dissipation rate = 120 Percentage of charge=0.7 Start value = 0 End value = 1	R=0.167480	This is an example of a working test case which uses **large input values.**
Inductance = 3.2 Capacitance = 6.3 Dissipation rate = 4.6 Percentage of charge=0.38 Start value = 0 End value = 2	R=0.692915	This is an example of a working test case which uses **in-between values of input**.

Inductance = 17.1 Capacitance = -8.8 Dissipation rate = -10.2 Percentage of charge=0.73 Start value = -1 End value = 1	The values of capacitance and dissipation rate are negative which is physically impossible. The program will prompt the user until he gives positive values.	This is an example of a test case where the values are physically impossible. **Tests invalid input values.**
Inductance = 100 Capacitance = 100 Dissipation rate = 100 Percentage of charge=0.4 Start value = 1 End value = 0	The end value given by the user is smaller than the end value, which doesn't make sense. The program will prompt the user until he gives the values in an appropriate order.	This is an example where the end and start values are in a wrong order. **Tests invalid input values.**
Inductance = 0.932 Capacitance = 0.324 Dissipation rate = 0.43 Percentage of charge = 0.64 Start value = -3 End value = 3	R=0.477539	This is an example of a working test case which uses **small input values.**
Inductance = 111 Capacitance = 291 Dissipation rate = 924 Percentage of charge = 0.4 Start value = 0 End value = 2	R=0.024170	This is an example of a working test case which uses **large input values.**
Inductance = 0 Capacitance = 0 Dissipation rate = 0 Percentage of charge = 0 Start value = 0 End value = 5	The values of L and C cannot be equal to 0 (because or else the program will attempt to divide by 0) and the value of the dissipation rate cannot be zero because the function would be a constant and thus no longer a function of x. However it's important to note that the	This is an example that tests extreme values: the zero values. **Tests invalid input values.**

	charge percentage can be equal to zero if we wanted.	
Selection of saved values	List of sets of values to select from. The user inputs a number to select the values he wants to do calculations with	Tests the selection of saved values
Saving data in the file	The program asks in which slot the user would like to save the set in the file	Tests saving data in the file
Invalid choices	The program will prompt the user to enter valid input for his choices. For example, when asked if he would like to save the values in the file, the user can answer ‘y’ or ‘n’. If he enters anything else, the program will correct him and display an error message.	Tests invalid input

3.2 Design

3.2.1 Introduction

The following program calculates the resistance in an RLC circuit by finding the roots of a function derived using Kirchhoff’s second law (Equation 3). A structure of type INPUT is defined at the beginning of the program, which contains 8 members. 4 of these include the input coefficients, and the other 4 are the following: the value of the root found (even though it is not an input value, it is still included in the structure to simplify the coding), the end value, the start value, and finally a variable which determines if the array has is available to take in new values, or if it is full.
We will use a structure array to save 5 sets of values of type INPUT.

The program will first attempt to open a file for reading, which is supposed to contain the input values. If it does not exist, it creates a new file and saves the structure array (with all elements in the array initialized to zero) into the file, then asks for the user for new input values. The user can choose to input up to 5 sets of values.

If the file exists (that is, the next time the user runs the program), it lists the contents of each element in the array. It then prompts the user to see if one of the existing values is to be used or to select new values, and the user will be prompted to save them in place of an existing set of values or in an element of the array which is available.

The program then calculates the left and right bounds of the function, then plots the function from its left to right boundaries, and prompts the user to select new start and end values to plot. It uses these start and end values to replot the function, and it also uses them in the bisection method for finding the resistance.

The program finally prints out the resistance found and asks the user if he wishes to save the values he used, and if he wishes to continue or stop.

3.2.2 Functions for Interacting with the User

- User-defined structure of type **INPUT**.
  Variables contained inside structure:
  - double **start** – start value for plotting
  - double **end** – end value for plotting
  - double **inductance** – inputted value of inductance
  - double **capacitance** – inputted value of capacitance
  - double **dissipationRate** – inputted value of the rate of dissipation
  - double **chargePercentage** – inputted value of the percentage of original charge
  - double **root** – outputted value of the root; we put this here to simplify to coding
  - int **full** – extra variable to notify the user whether the array is full or not; can take TRUE (1) or FALSE (0) values.

- **void main()**
  Logic/Algorithm
  This function controls the rest of the program. It first defines the following variables to save desired values:
  - **input[NUM_VALUES]** – array of type INPUT. Can contain NUM_VALUES values, which we will define as a symbolic constant to be equal to 5.
  - **newSet** – structure variable of type INPUT. Used to take input values from the user.
  - **choice** – character which depends on the answers of the users. Can take either 'y' or 'n' depending on the user's choice.
  - **save** – character which can take either 'y' or 'n' depending on whether the user wants to save values in a file or not.
  - **slot** – integer for which the user can pick any number from 1 to 5 after being prompted which slot in the file he would like to save his values in.

- **set** – integer number for which the user can pick any number from 1 to 5 after being prompted which set he would like to use.
- **flag** – flag used to repeat certain loops in case of invalid input.
- **answer** – integer number which can take values 1 or 2 depending on the user's choice.
- **ix** – index used for going through the five values of the array.

The main function starts off by calling **readBin**, to check whether the file exists or not.

If it does not, in other words if input[ix].full == FALSE (ix goes from 0 to 4, here we ue a for loop), then it asks the user for new input values by calling **userInput** and then plots the function, and then searches for the root using the function **bisectionSearch.** The main function repeatedly saves the input values in the file by calling the function **saveInFile**. Inside this if condition, there is a do-while loop, with conditions while(ix<NUM_VALUES && choice=='y'), to make sure that the loop does not go beyond the contents of the array and also depending on whether the user enters 'y' for yes.

In the case where the file exists (we can use an "else" condition here), so the next time the user runs the program, the main function calls **displayFile** to display the values contained in the file and asks the user whether he would like to use one of the pre-existing sets or if he would like to input new values (he needs to enter a number, either 1 or to, for "answer").
If he chooses to use the pre-existing sets (if answer==1), he is prompted to select which one (a number from 1 to 5 is to be entered) and then the main function calls **choiceInput**.

If he chooses to enter new values (if answer==2), then the main repeats the algorithm of the first part. He is then asked if he would like to save the values, and in which slot in the file.

Using a do-while loop, with condition while(choice=='y'), he may choose to repeat this second part if he wishes, in which cases he is again prompted whether he wants to use a pre-existing set of values or enter his own new values.

Multiple flags play a role in this part, making sure the user enters proper values and characters for each choice he makes. Multiple while loops correct the user if he enters invalid input.

- **void displayFile(INPUT *)**

Parameters

- **inputPtr –** this variable contains the address of the array of type INPUT.

Logic/algorithm

This function displays the current contents of the inputted values in the array, to which inputPtr points. These are actually the values currently contained in the file. To do this, we use a for loop.

- **void userInput(INPUT *)**

  Parameters

  - **inputPtr –** this variable contains the address of the array of type INPUT.

  Logic/algorithm

  The function prompts the user for capacitance, inductance, dissipation rate and charge percentage. It then calls a function **computeFuncBounds** to compute the upper and lower bounds of the function using the inputted coefficients. After this, it calls the function **computeDataPoints** to fill the array of the function and to finally plot the graph of the function. Finally the function asks the user to input appropriate start and end values for the plotting and for calculating the root, and the main function will these use these in order

  Loops/conditions

  - **while –** loops in case the user inputs invalid values; for example a bigger start value than end value, or negative coefficients (negative capacitance, inductance…), or even if the user inputs start and end values outside the domain of the function.

- **void choiceInput(INPUT *, int *)**

  Parameters

  - **inputPtr –** this variable contains the address of the array of type INPUT.

  Logic/algorithm

  Similar to the userInput function. This time, the number of the set is passed and this function uses one of the preexisting sets in the file to calculate the root and plot the function, so the user does not have to input the electrical properties of the circuit, but only the start and end values for replotting. It uses inputPtr[*set-1].start for the start value for example, because if the user enters 1, this function will store the values entered in the 0th cell of the array.

3.2.3 Functions for Calculations

- **double computeFuncBounds(INPUT *)**

  Parameters

  - **inputPtr** – this variable contains the address of the array of type INPUT. This variable contains the majority of the values used in the program.

  Return value

  This function returns the absolute value of the bounds of the function in Equation (3).

  Logic/algorithm

  This function defines a local variable **bound** of type double and computes the bound of the function. This is obtained mathematically by isolating what's inside the square root term of Equation (3) and setting it superior to 0 (a negative square root is imaginary): this will give us information about the interval on which the function is defined. Doing the math, it turns out that $-2\sqrt{\frac{L}{C}} < R < 2\sqrt{\frac{L}{C}}$. Thus this is the domain of the function.

- **double func(INPUT *, double)**

  Parameters

  - **inputPtr** – this variable contains the address of the array of type INPUT. This variable contains the majority of the values used in the program.
  - **x** – variable of type double. This is the x-component of the function in Equation 3. x here is actually, normally, equivalent to the resistance. Later on, we will attempt to calculate the value of that x when the function of Equation (3) is set to 0.

  Return values

  The function returns **fx**, which is a variable of type double declared in this function. This is equivalent to the y-component of the function in Equation (3) and is a function of **x,** or the resistance. Generally, this fx is set equal to 0 in order to find its root x.

  Logic/algorithm

  The function defines a variable **fx** of type double which is a function of the parameter **x**. What this function actually does is break down Equation (3) line by line.

- **double bisectionSearch(INPUT *, double *)**

  Parameters

- **inputPtr** – this variable contains the address of the array of type INPUT. This variable contains the majority of the values used in the program.
- **root** – points to the root member of the variable of type INPUT defined in the main function. This parameter replaces the value of **root** in the structure with an approximation of the root using the bisection method, at the end of the function.

Return values

- **root** – returns the updated value of the root.

Logic/algorithm

The function first defines a variable **est** of type double which is equivalent to the estimate of the root after each iteration, and a variable **flag** of type int which simply acts as a "marker" for the later discussed do-while loop. **flag** can take values TRUE (1) or FALSE (0) and depending on the value it is taking it either stops the while loop or allows it to keep running.

As perhaps one of the most important functions in this code, it uses the bisection method to converge the value of **est** to a value closer and closer to the root of the function, which we have actually defined in the function **func**. The method consists of repeatedly dividing an interval previously chosen by the user in the function **userInput** (start and end values) by half, until the absolute value of f(left bound)*f(right bound) is smaller than some tolerance value (which can be considered equal to 0).

If either of the intervals contains the root, it updates either the left or right bound accordingly to the calculated value of the estimate.

Multiple calls to **func** will allow us to calculate the value of f(left bound) and f(right bound).

Loops/conditions

- **do-while (flag==FALSE)** – the value of flag is only updated once the function converges to an acceptable value of the root (we know this with the help of the tolerance value). Until then, its value is FALSE.
- **if conditions** – changes the value of either the left or right bound depending on whether the root is located in the left or right interval.

3.2.4 Functions for Plotting

- **void computeDataPoints(INPUT *)**

Parameters

- **inputPtr** – this variable contains the address of the array of type INPUT. This variable contains the majority of the values used in the program.

Logic/algorithm

The point of this function is to simply fill in the values of a 2-dimensional array **points[2][NUM_POINTS]**, with row 0 being the100 values of the x-component and the row 1 being the 100 values of the y-component, to later be able to plot the function given enough points.

It starts off with an increment **inc** of type double equal to (end-start)/NUM_POINTS (NUM_POINTS is a symbolic constant set to be 100, it is the number of points used to plot), and going from the x and y-components of the start value, uses a **for loop (for ix=1; ix<NUM_POINTS; ix++)** to fill in the array with 100 points going from [1] to [NUM_POINTS-1] (or 99) (it starts at 1 because the 0th value is simply the start value defined by the user).

Finally, this function calls the function **plot** to which it passes the values of the filled array points in order to actually plot the graph.

- **void plot(INPUT *, double [][NUM_POINTS])**

  Parameters

  - **inputPtr** – this variable contains the address of the array of type INPUT. This variable contains the majority of the values used in the program.
  - **points[][NUM_POINTS]** – these are all the updated values of the points going from the start to end bound (either chosen by the user or simply calculated by **computeFuncBounds**, depending where we are in the program) used to plot the function.

  Logic/Algorithm

  This function plots the graph of the function of Equation (3). It receives the calculated values for the x and y axis from the two dimensional array **points[][NUM_POINTS**].

  It calls the functions **getMin** and **getMax** to calculate the extreme values of the range of the function to be plotted.

  Finally, if a root is to be found, it draws an “x” on the graph at this point.

  With the call to different functions in the PLplot library, we can change different visual parameters of the plotted graph.

  Loops/conditions

  - **if-condition** – if a root is found, draws an “x” on the graph at this point.

- **double getMin(double [])**

  Parameters

  - **arr[]** – Pointer to the y-component of the previous **points[][NUM_POINT]** which contains all the values for plotting.

Returned values

- **min** – returns the value of the minimum.

Logic/algorithm

The function traverses the array in order to find minimum value. At the end, the function returns the minimum value found in the array.

Loops/conditions

- **for(ix=1; ix<NUM_POINTS; ix++)** – goes over every single term of the array and if any of the values is inferior to **min** it replaces the min with arr[ix].

- **double getMax(double [])**

  Parameters

  - **arr[]** – Pointer to the y-component of the previous **points[][NUM_POINT]** which contains all the values for plotting.

  Returned values

  - **max** – returns the value of the maximum.

  Logic/algorithm

  The function traverses the array in order to find maximum value. At the end, the function returns the maximum value found in the array.

  Loops/conditions

  - **for(ix=1; ix<NUM_POINTS; ix++)** – goes over every single term of the array and if any of the values is superior to **max** it replaces the max with arr[ix].

3.2.5 Functions for Files

- **void saveInFile(INPUT *, input *, int *)**

  Parameters

  - **inputPtr** – this variable contains the address of the structure of type INPUT. Points to newSet
  - **values[]** – points to input in the main. Contains the constantly updated sets.
  - **int** – points to either the slot in which the user wants to save the values, or to the set that the user is trying to use to compute the bounds, depending where he is in the program.

  Logic/algorithm

This function saves the ix'th set in its own slot in the array. Then, it calls **writeBin** to write these values in the file and passes the array values.

- **void readBin(INPUT *)**

  Parameters

  - **values[] –** contains the constantly updated sets of values.

  Logic/algorithm

  Reads from the binary file and checks whether the file exists or not.
  This function creates a pointer filePtr which points to the file BIN_FILE defined in the symbolic constants as "projectdata.bin". It then opens it using the fopen function, in mode "rb".
  Using two if conditions:

  - if filePtr==NULL (if the file does not exist), then this function assigns, using a for loop, the value FALSE to values[ix], 0<ix<4.
  - else (if the file exists) then the loop does the same thing but instead of assigning the value FALSE, it assigns the value TRUE. The file is then closed using fclose.

- **void writeBin(INPUT *)**

  Parameters

  - **values[] –** contains the constantly updated sets of values.

  Logic/algorithm

  Writes in the binary file. Takes the values from the function **saveInFile** to which values[] points to, and writes them in this file. The file is first opened with the "wb" mode; then using two if conditions:

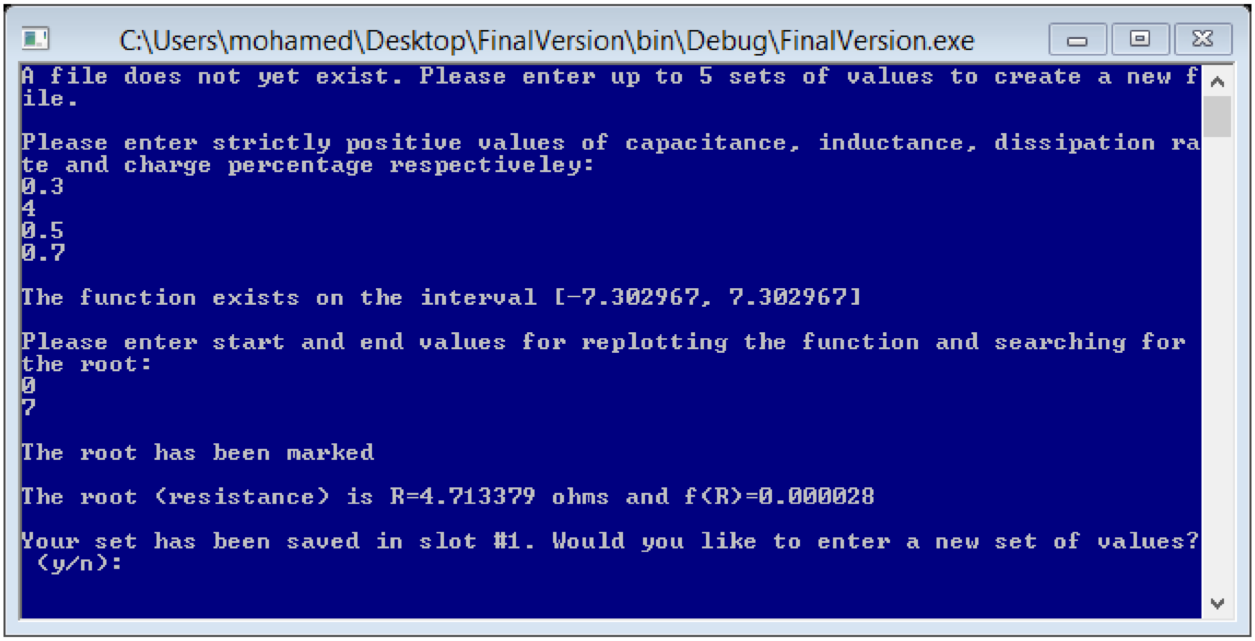
  - if filePtr==NULL, then this means there is actually an error opening the file. A message will be displayed.
  - else, the file is opened correctly, then the function fwrite is used to write the array values[] in the file. The file is then closed using fclose.

4 Implementation

See the program in FinalVersion.c.

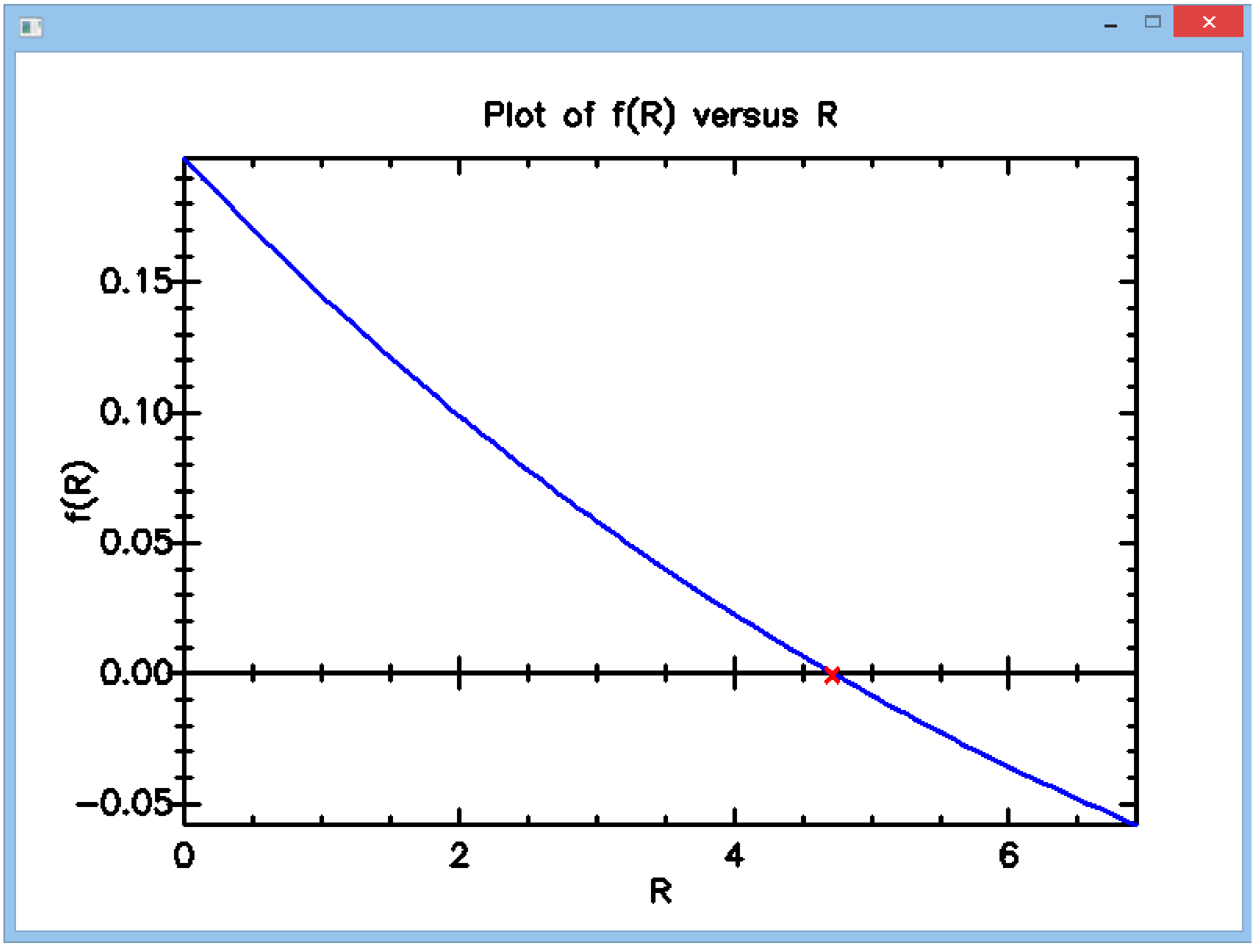
5 Software Testing and Verification

- Test case 1

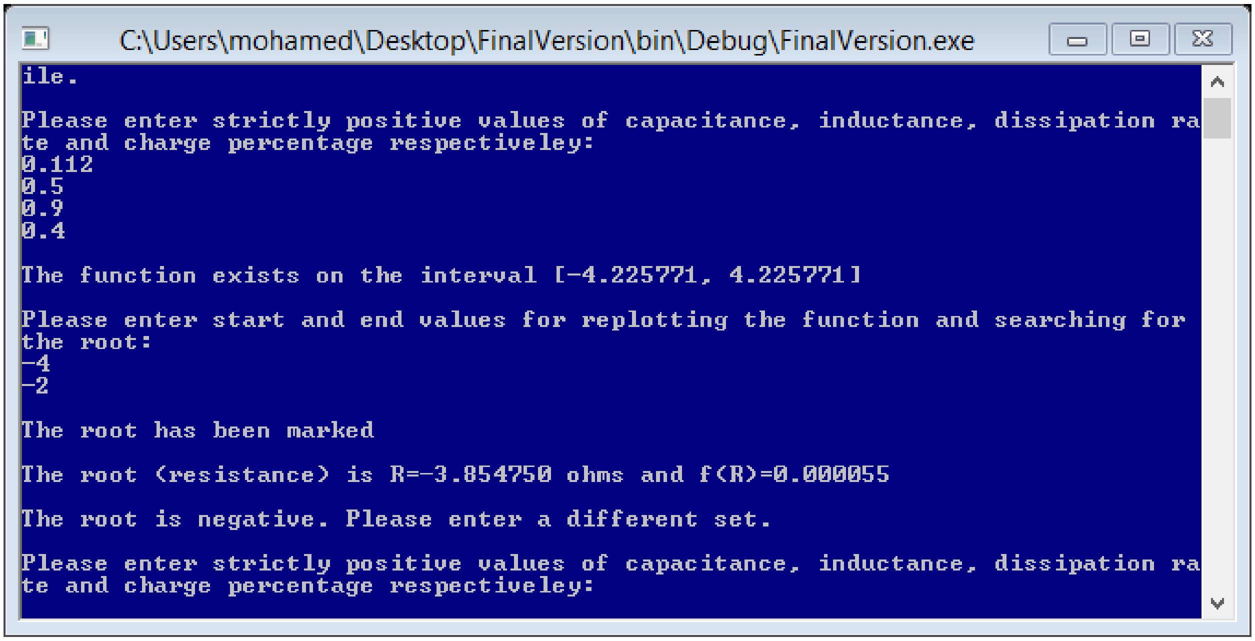


C:\Users\mohamed\Desktop\FinalVersion\bin\Debug\FinalVersion.exe

```
A file does not yet exist. Please enter up to 5 sets of values to create a new f
ile.

Please enter strictly positive values of capacitance, inductance, dissipation ra
te and charge percentage respectiveley:
0.3
4
0.5
0.7

The function exists on the interval [-7.302967, 7.302967]

Please enter start and end values for replotting the function and searching for
the root:
0
7

The root has been marked

The root (resistance) is R=4.713379 ohms and f(R)=0.000028

Your set has been saved in slot #1. Would you like to enter a new set of values?
 (y/n):
```

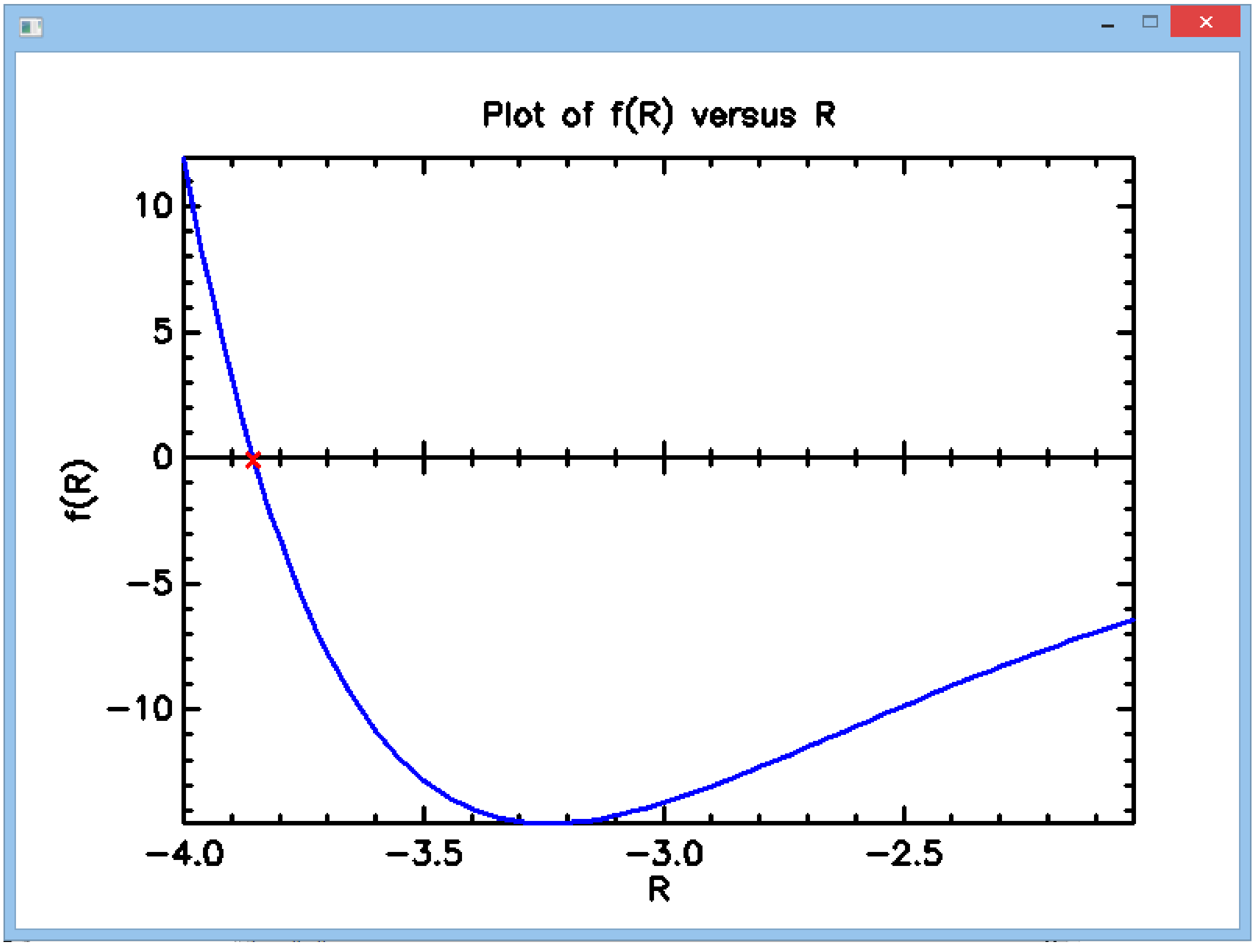


- Test case 2

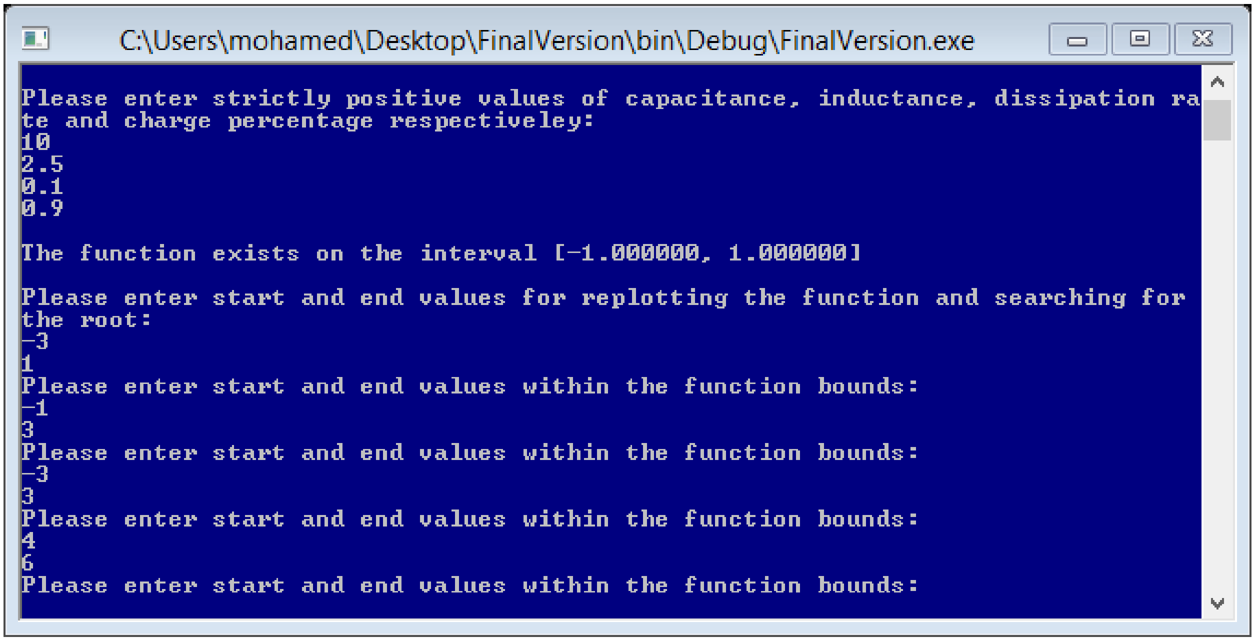


C:\Users\mohamed\Desktop\FinalVersion\bin\Debug\FinalVersion.exe

```
ile.

Please enter strictly positive values of capacitance, inductance, dissipation ra
te and charge percentage respectiveley:
0.112
0.5
0.9
0.4

The function exists on the interval [-4.225771, 4.225771]

Please enter start and end values for replotting the function and searching for
the root:
-4
-2

The root has been marked

The root (resistance) is R=-3.854750 ohms and f(R)=0.000055

The root is negative. Please enter a different set.

Please enter strictly positive values of capacitance, inductance, dissipation ra
te and charge percentage respectiveley:
```

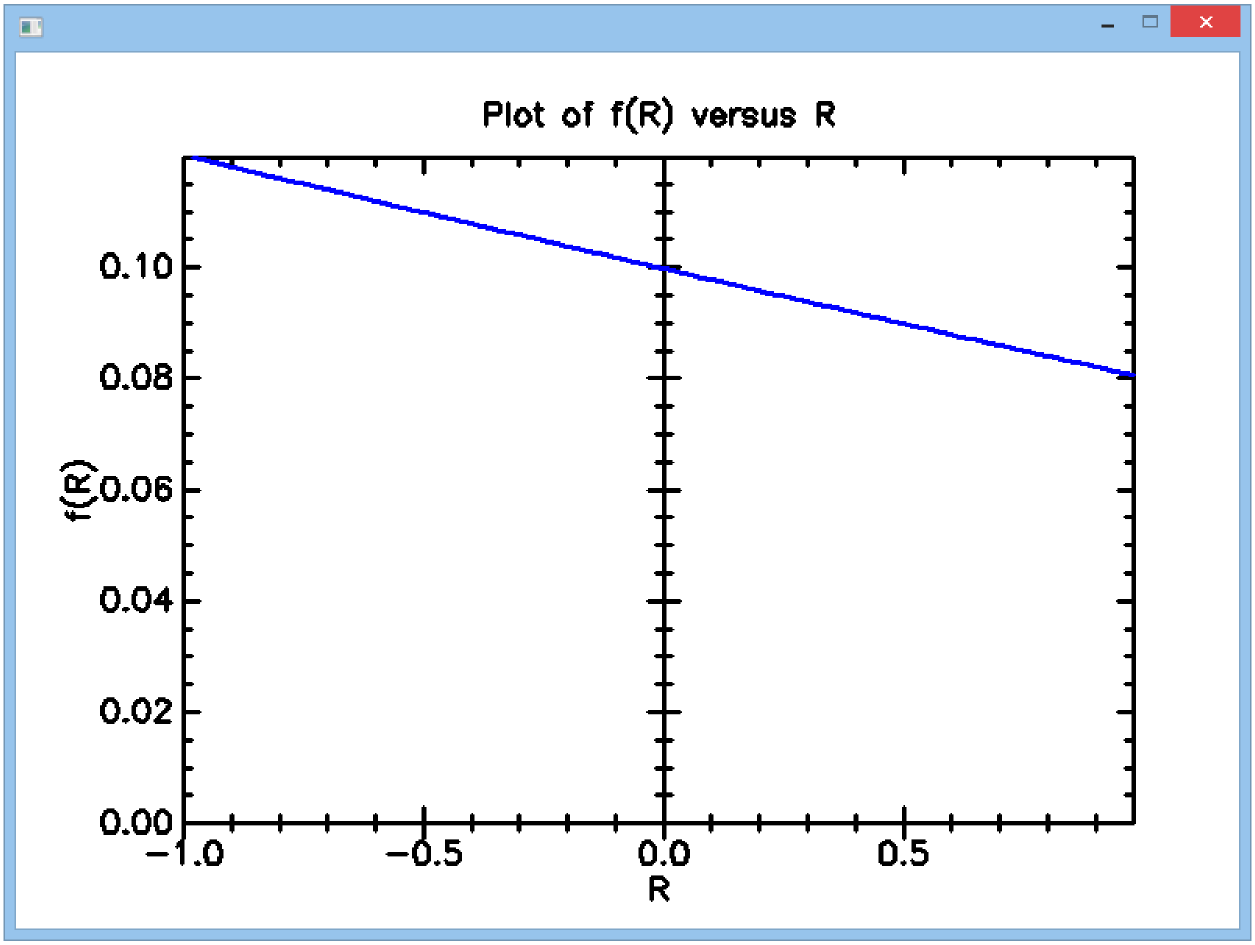


- Test case 3

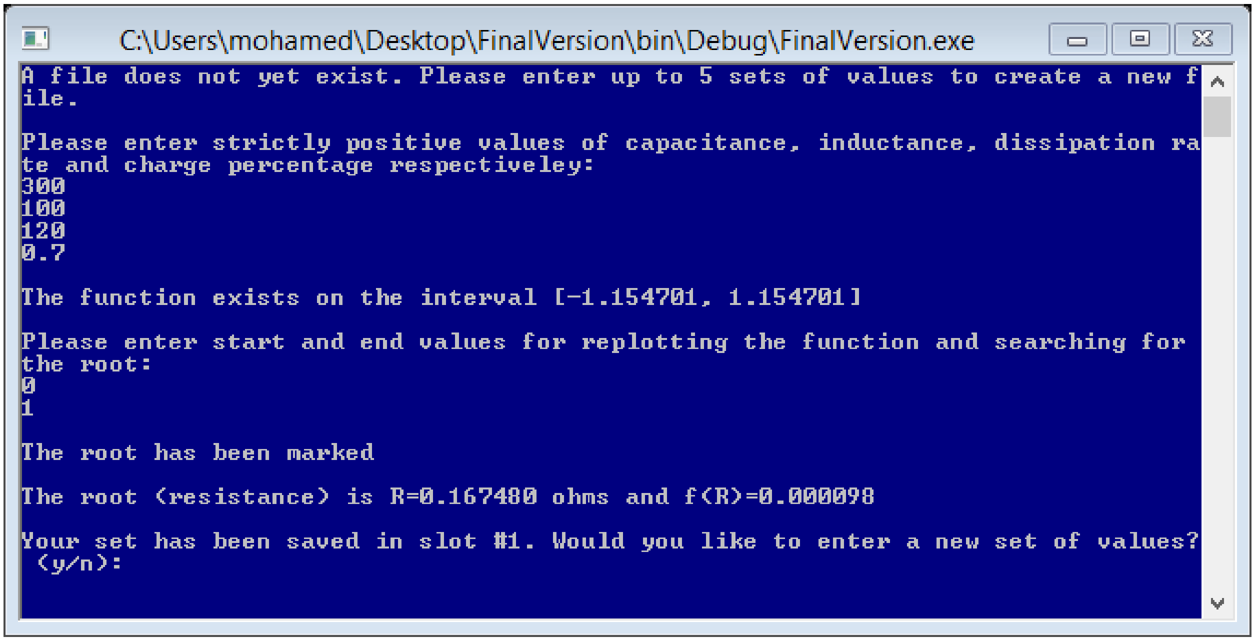


C:\Users\mohamed\Desktop\FinalVersion\bin\Debug\FinalVersion.exe

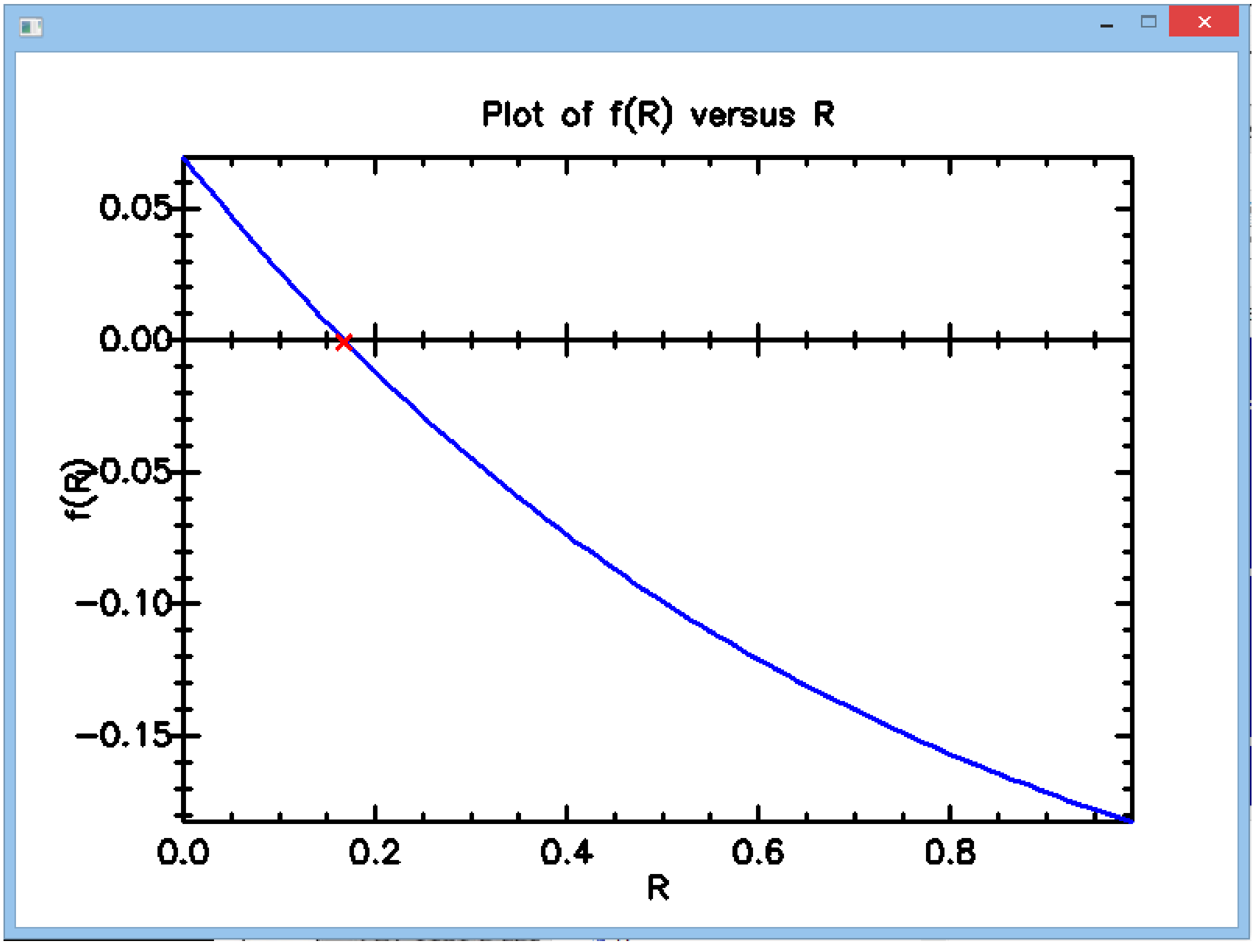
```
Please enter strictly positive values of capacitance, inductance, dissipation ra
te and charge percentage respectiveley:
10
2.5
0.1
0.9

The function exists on the interval [-1.000000, 1.000000]

Please enter start and end values for replotting the function and searching for
the root:
-3
1
Please enter start and end values within the function bounds:
-1
3
Please enter start and end values within the function bounds:
-3
3
Please enter start and end values within the function bounds:
4
6
Please enter start and end values within the function bounds:
```



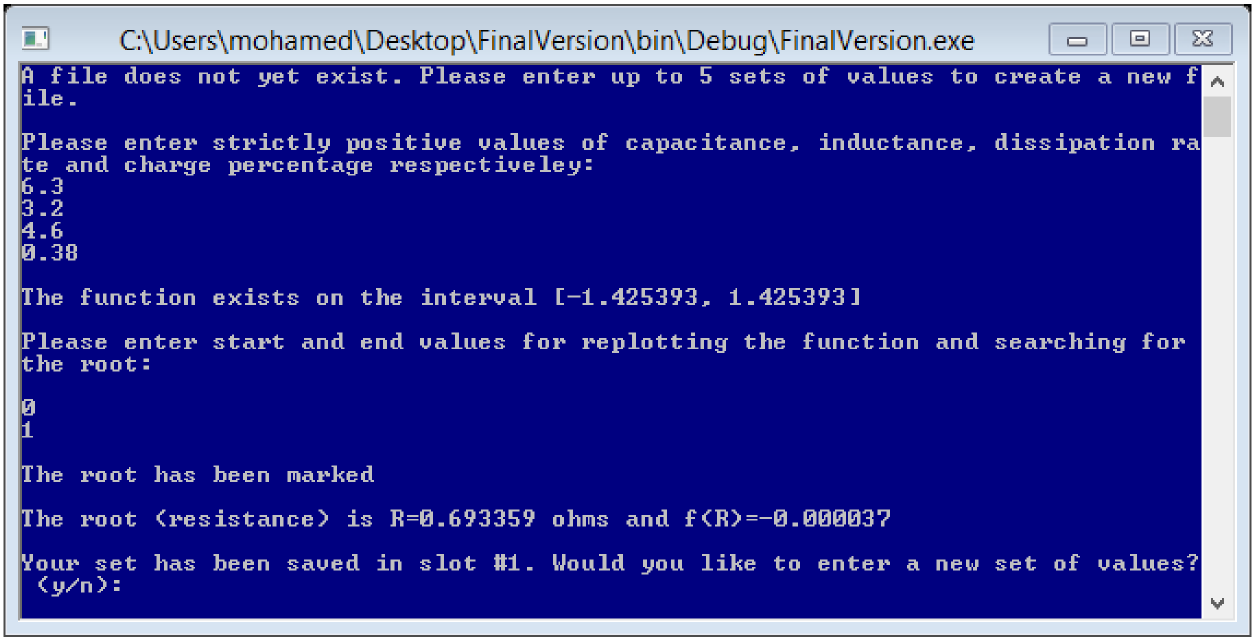
- Test case 4


C:\Users\mohamed\Desktop\FinalVersion\bin\Debug\FinalVersion.exe

```
A file does not yet exist. Please enter up to 5 sets of values to create a new f
ile.

Please enter strictly positive values of capacitance, inductance, dissipation ra
te and charge percentage respectiveley:
300
100
120
0.7

The function exists on the interval [-1.154701, 1.154701]

Please enter start and end values for replotting the function and searching for
the root:
0
1

The root has been marked

The root (resistance) is R=0.167480 ohms and f(R)=0.000098

Your set has been saved in slot #1. Would you like to enter a new set of values?
 (y/n):
```

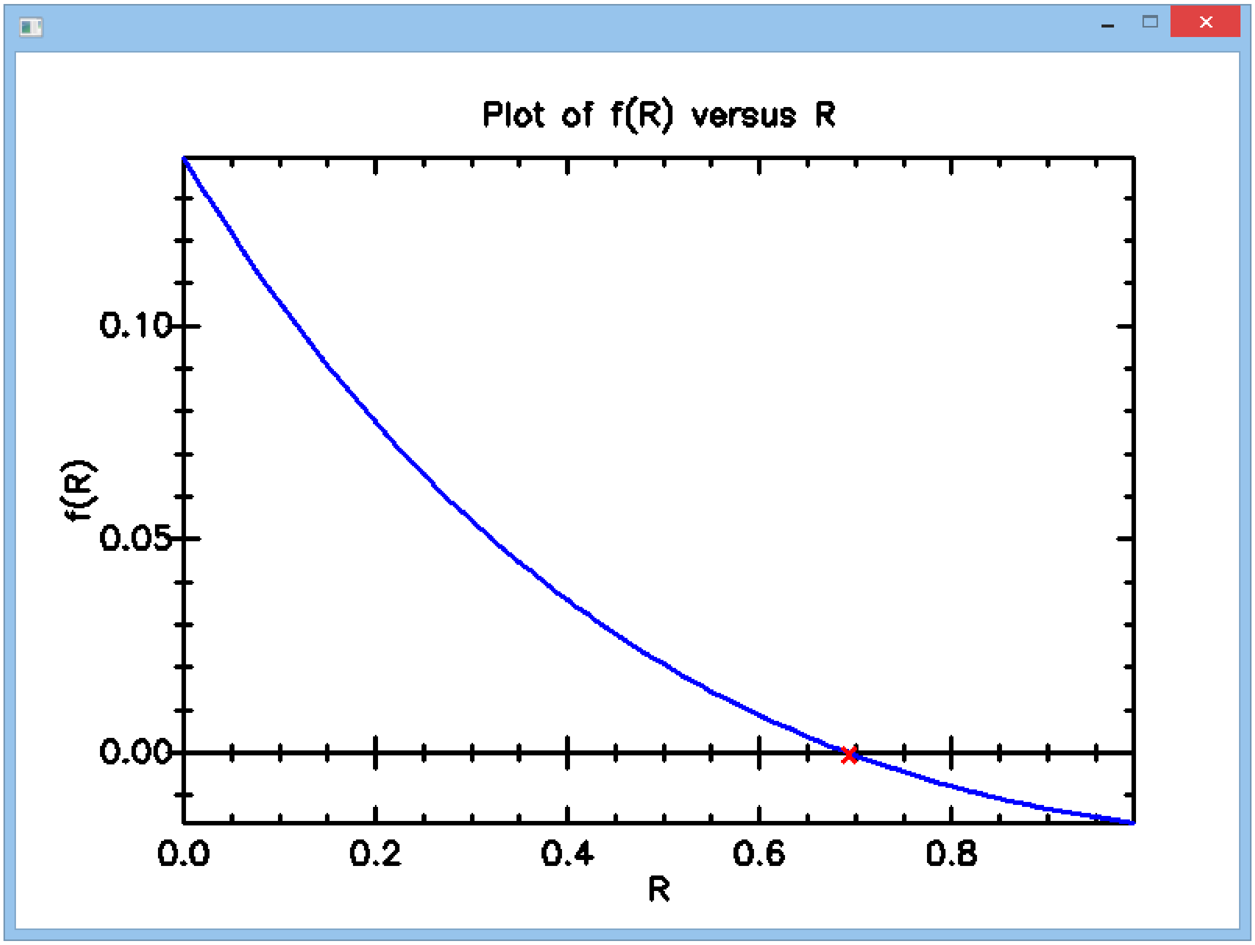


- Test case 5

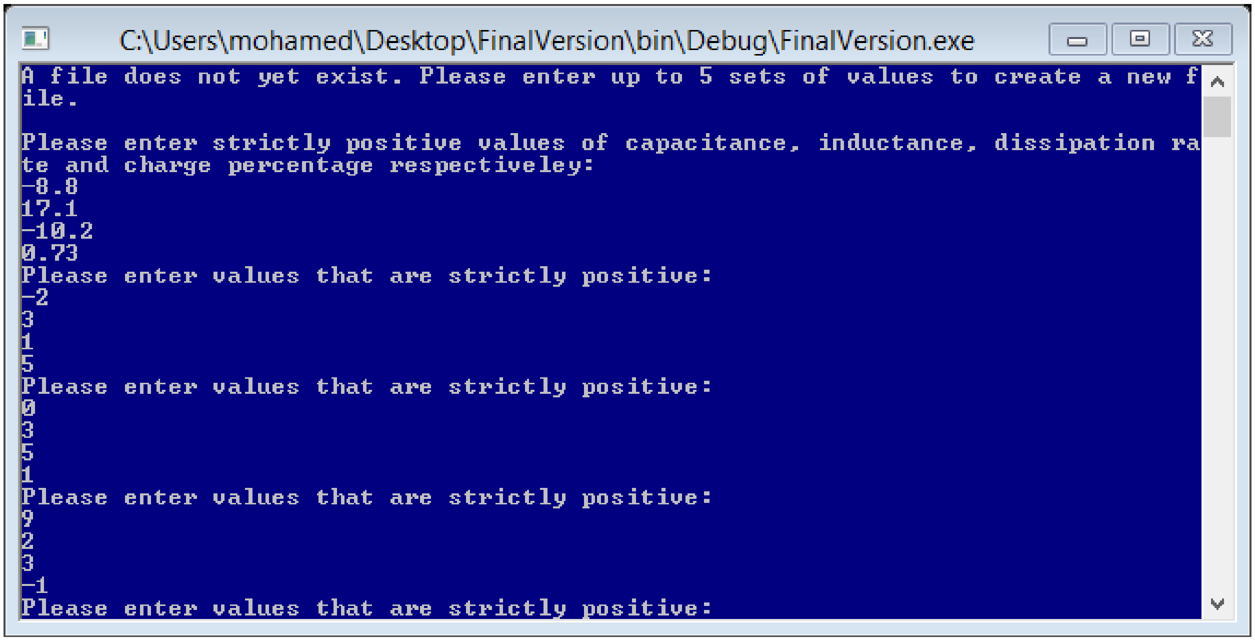


C:\Users\mohamed\Desktop\FinalVersion\bin\Debug\FinalVersion.exe

```
A file does not yet exist. Please enter up to 5 sets of values to create a new f
ile.

Please enter strictly positive values of capacitance, inductance, dissipation ra
te and charge percentage respectiveley:
6.3
3.2
4.6
0.38

The function exists on the interval [-1.425393, 1.425393]

Please enter start and end values for replotting the function and searching for
 the root:

0
1

The root has been marked

The root (resistance) is R=0.693359 ohms and f(R)=-0.000037

Your set has been saved in slot #1. Would you like to enter a new set of values?
 (y/n):
```

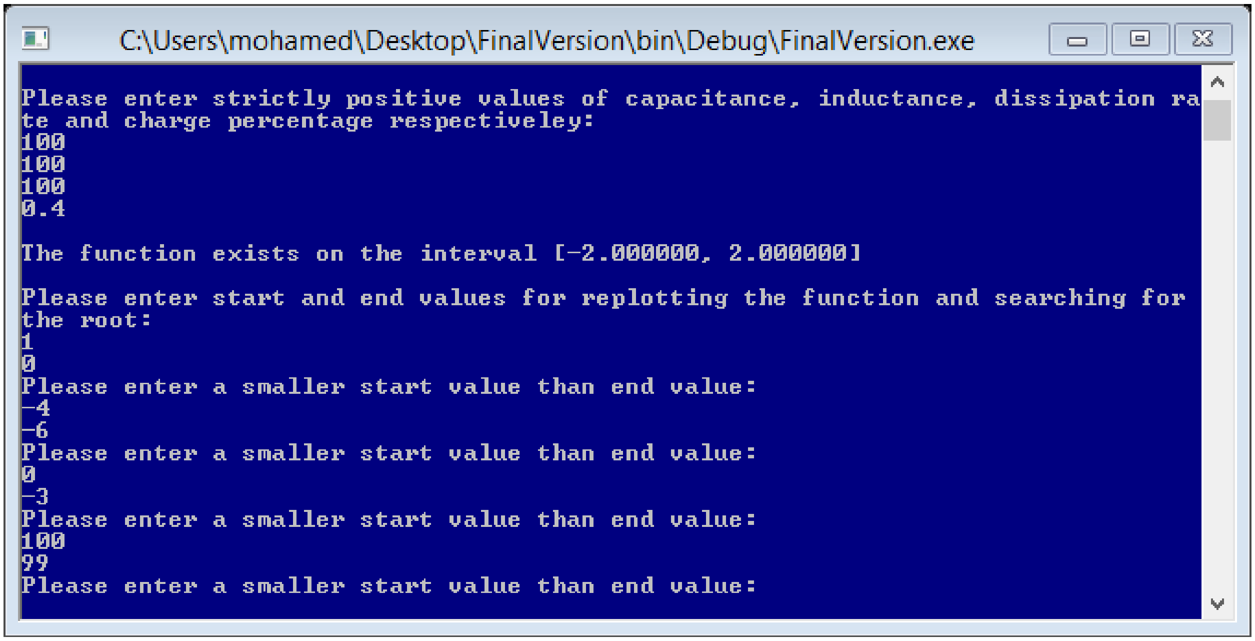


- Test case 6


C:\Users\mohamed\Desktop\FinalVersion\bin\Debug\FinalVersion.exe

```
A file does not yet exist. Please enter up to 5 sets of values to create a new f
ile.

Please enter strictly positive values of capacitance, inductance, dissipation ra
te and charge percentage respectiveley:
-8.8
17.1
-10.2
0.73
Please enter values that are strictly positive:
-2
3
1
5
Please enter values that are strictly positive:
0
3
5
1
Please enter values that are strictly positive:
9
2
3
-1
Please enter values that are strictly positive:
```

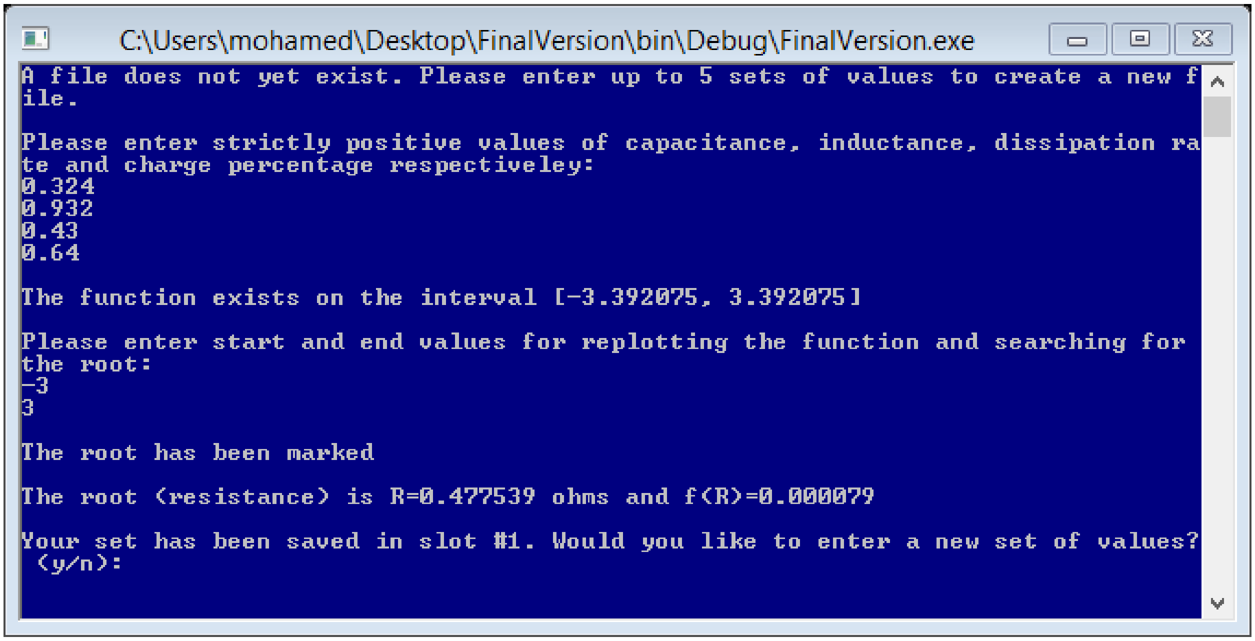
- Test case 7



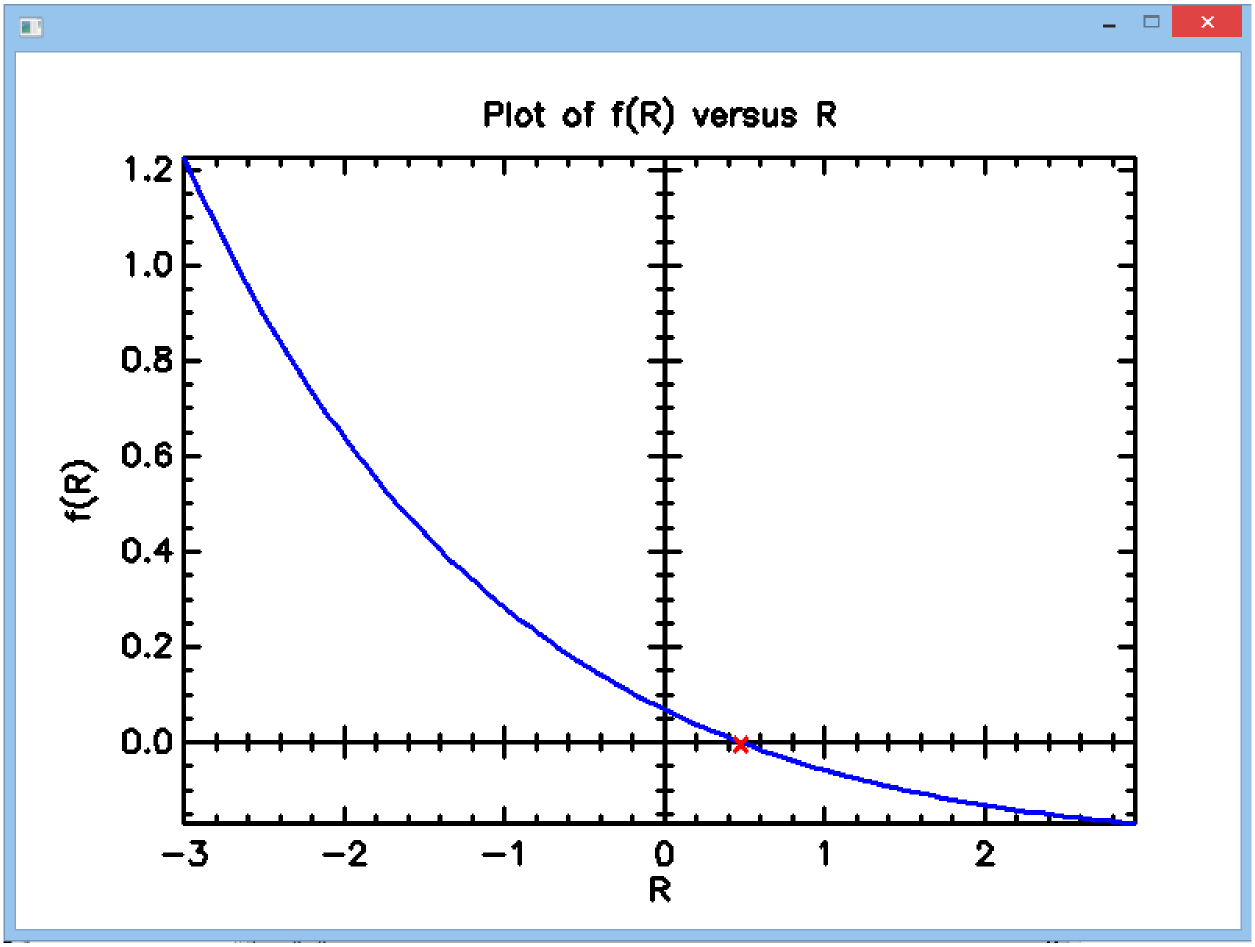
C:\Users\mohamed\Desktop\FinalVersion\bin\Debug\FinalVersion.exe

```
Please enter strictly positive values of capacitance, inductance, dissipation ra
te and charge percentage respectiveley:
100
100
100
0.4

The function exists on the interval [-2.000000, 2.000000]

Please enter start and end values for replotting the function and searching for
the root:
1
0
Please enter a smaller start value than end value:
-4
-6
Please enter a smaller start value than end value:
0
-3
Please enter a smaller start value than end value:
100
99
Please enter a smaller start value than end value:
```

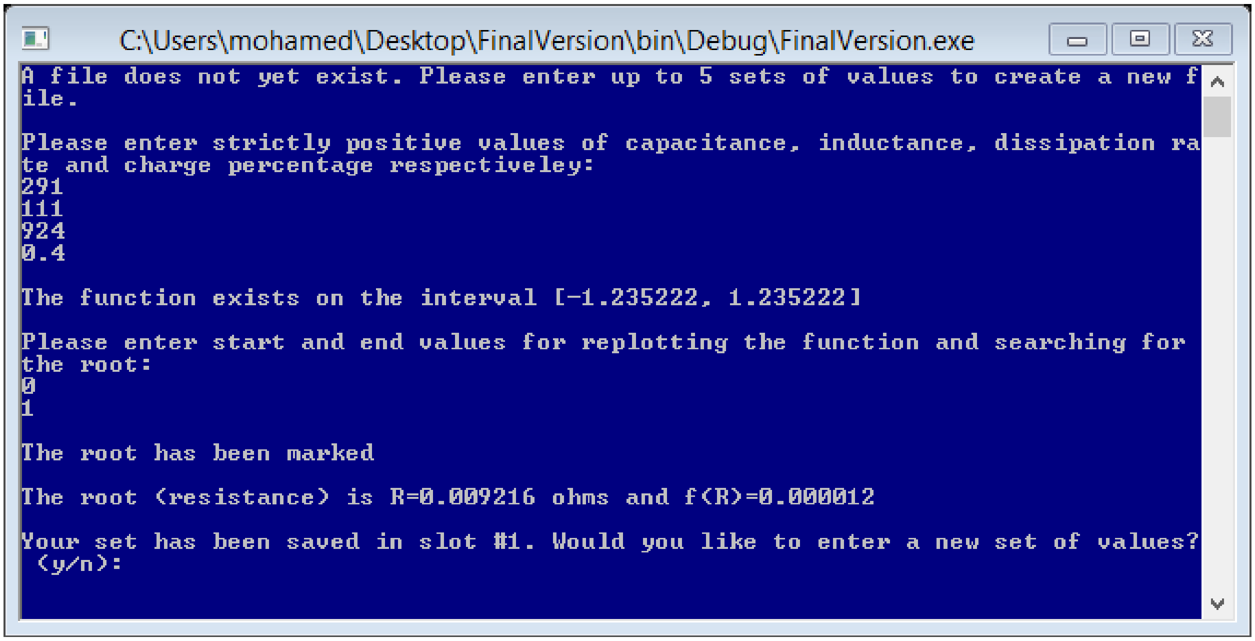
- Test case 8


C:\Users\mohamed\Desktop\FinalVersion\bin\Debug\FinalVersion.exe

```
A file does not yet exist. Please enter up to 5 sets of values to create a new f
ile.

Please enter strictly positive values of capacitance, inductance, dissipation ra
te and charge percentage respectiveley:
0.324
0.932
0.43
0.64

The function exists on the interval [-3.392075, 3.392075]

Please enter start and end values for replotting the function and searching for
the root:
-3
3

The root has been marked

The root (resistance) is R=0.477539 ohms and f(R)=0.000079

Your set has been saved in slot #1. Would you like to enter a new set of values?
 (y/n):
```

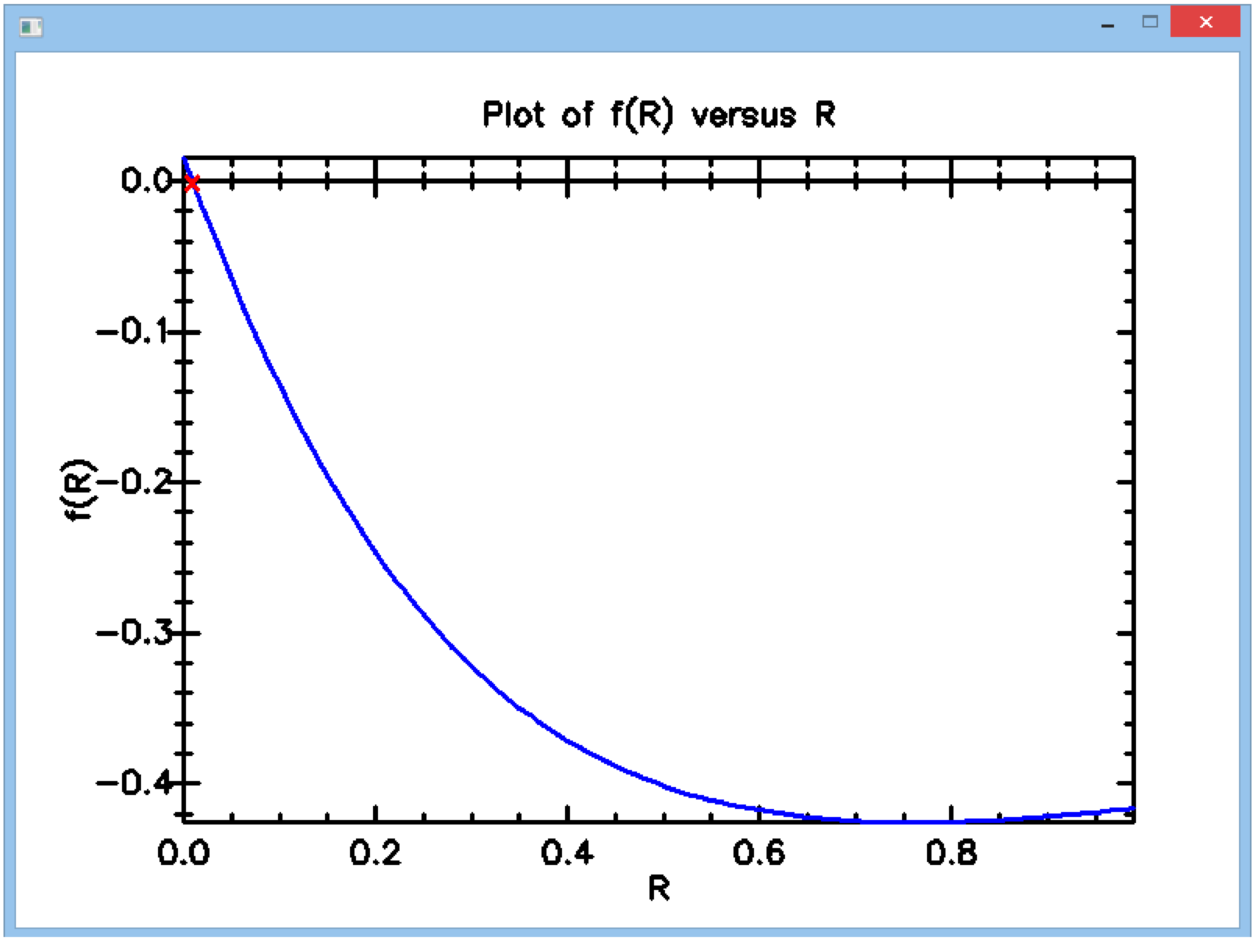


- Test case 9

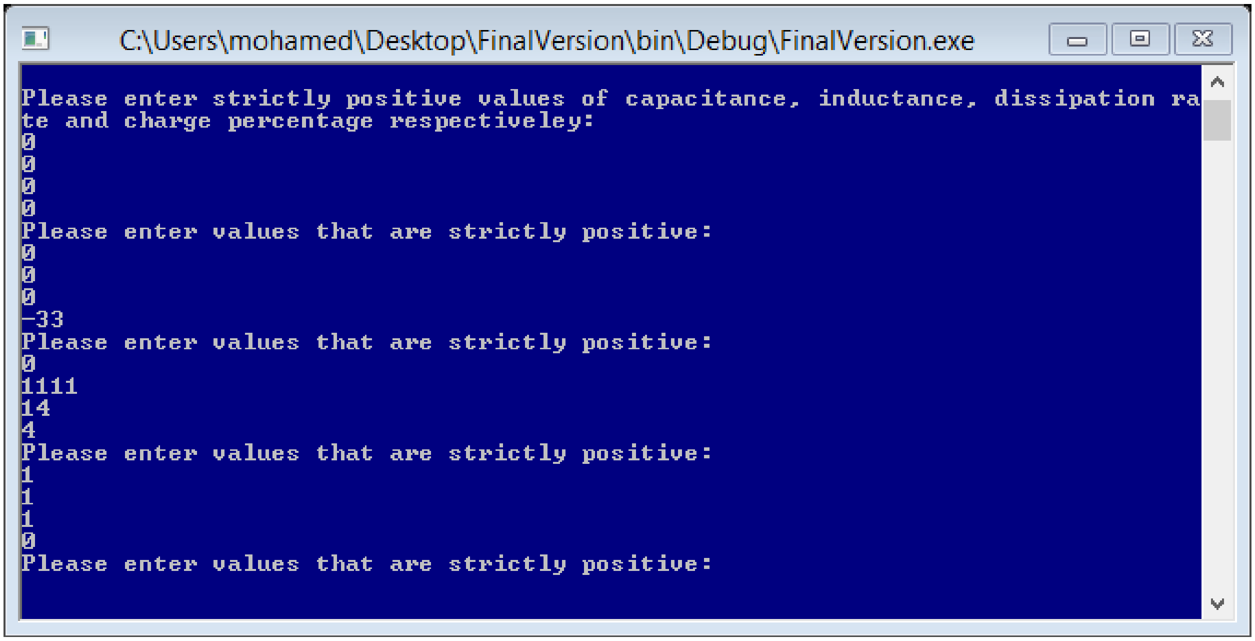


C:\Users\mohamed\Desktop\FinalVersion\bin\Debug\FinalVersion.exe

```
A file does not yet exist. Please enter up to 5 sets of values to create a new f
ile.

Please enter strictly positive values of capacitance, inductance, dissipation ra
te and charge percentage respectiveley:
291
111
924
0.4

The function exists on the interval [-1.235222, 1.235222]

Please enter start and end values for replotting the function and searching for
the root:
0
1

The root has been marked

The root (resistance) is R=0.009216 ohms and f(R)=0.000012

Your set has been saved in slot #1. Would you like to enter a new set of values?
 (y/n):
```



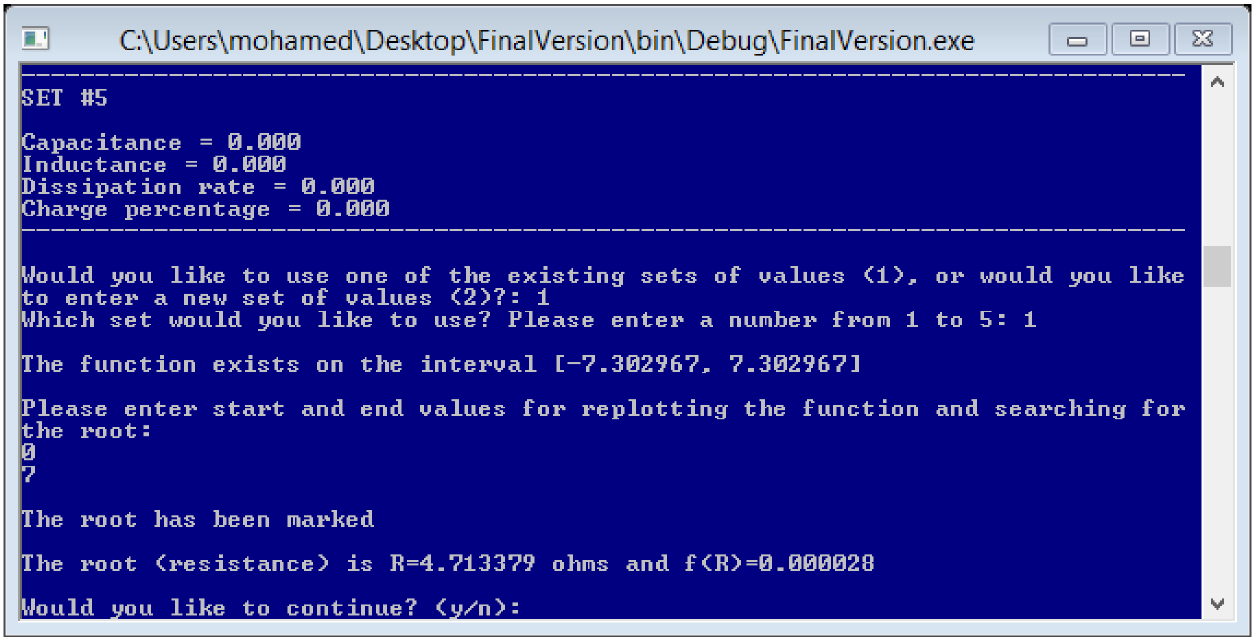
- Test case 10



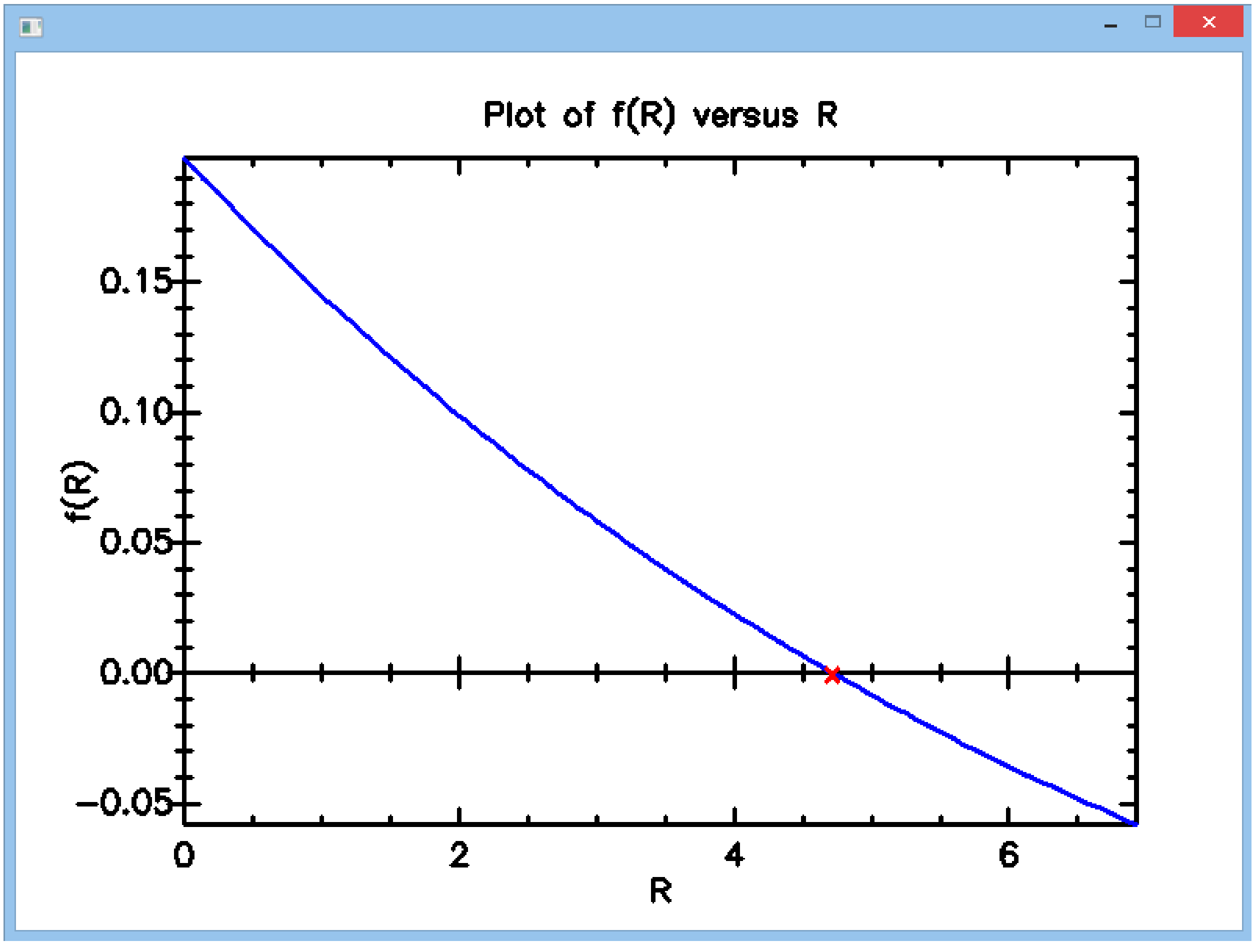
C:\Users\mohamed\Desktop\FinalVersion\bin\Debug\FinalVersion.exe

```
Please enter strictly positive values of capacitance, inductance, dissipation ra
te and charge percentage respectiveley:
0
0
0
0
Please enter values that are strictly positive:
0
0
0
-33
Please enter values that are strictly positive:
0
1111
14
4
Please enter values that are strictly positive:
1
1
1
0
Please enter values that are strictly positive:
```

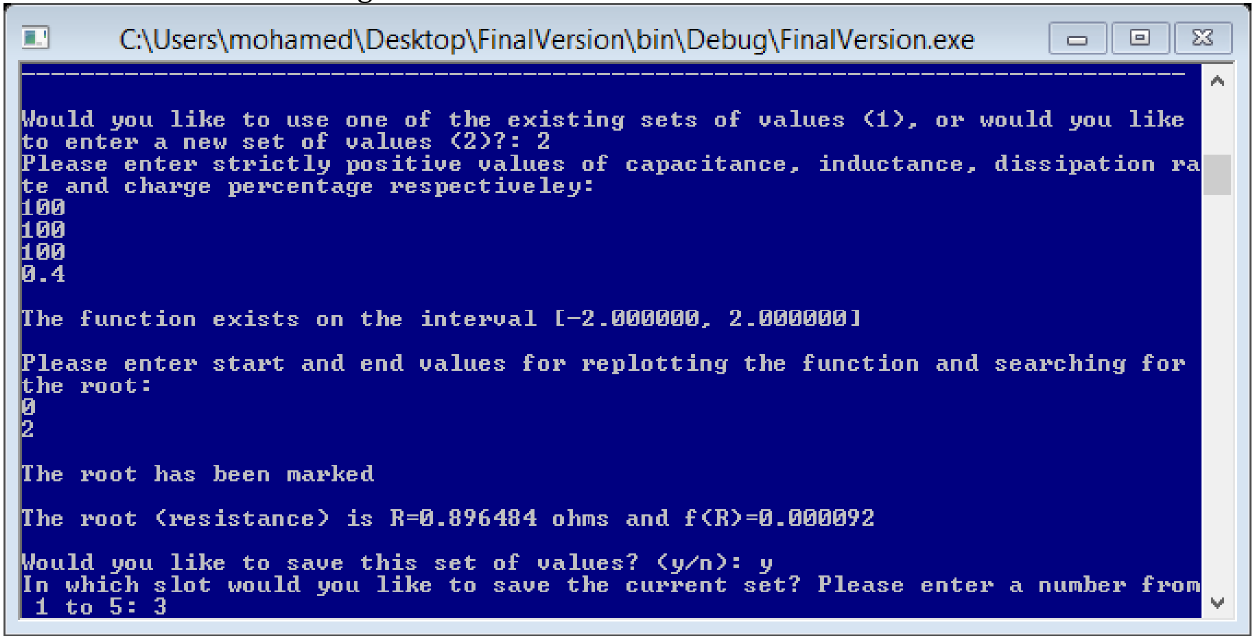
- Test case 11: Selection of saved values


C:\Users\mohamed\Desktop\FinalVersion\bin\Debug\FinalVersion.exe

```
--------------------------------------------------------------------------------
SET #5

Capacitance = 0.000
Inductance = 0.000
Dissipation rate = 0.000
Charge percentage = 0.000
--------------------------------------------------------------------------------

Would you like to use one of the existing sets of values (1), or would you like
to enter a new set of values (2)?: 1
Which set would you like to use? Please enter a number from 1 to 5: 1

The function exists on the interval [-7.302967, 7.302967]

Please enter start and end values for replotting the function and searching for
the root:
0
7

The root has been marked

The root (resistance) is R=4.713379 ohms and f(R)=0.000028

Would you like to continue? (y/n):
```

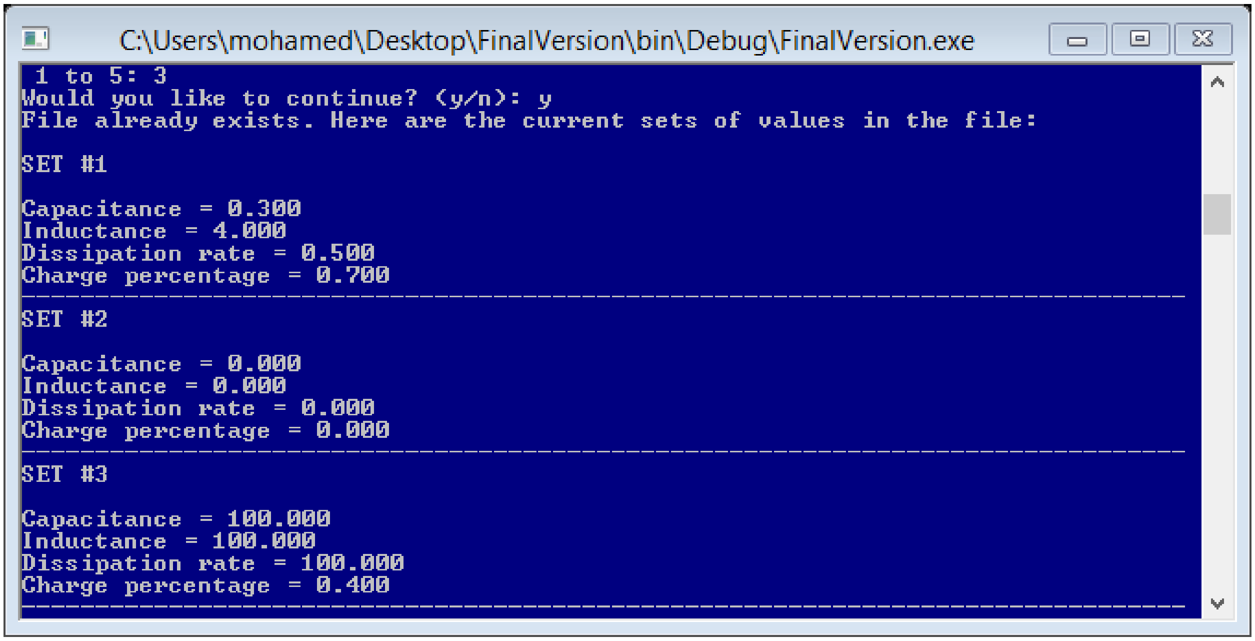


- Test case 12: Saving data into the file


C:\Users\mohamed\Desktop\FinalVersion\bin\Debug\FinalVersion.exe

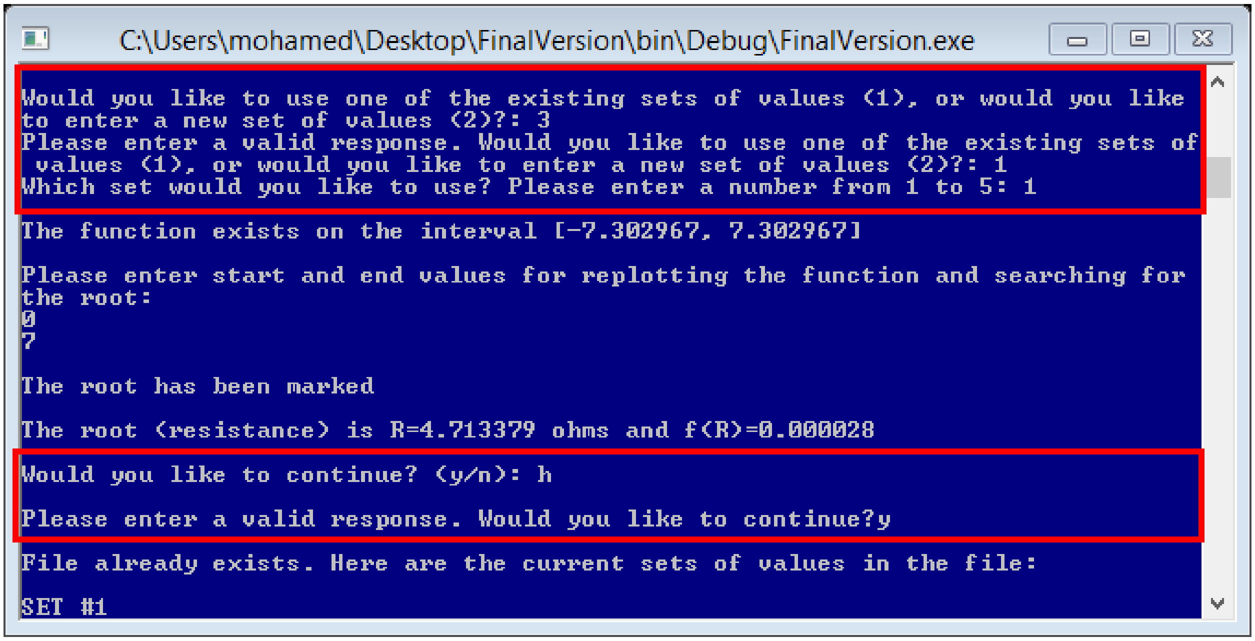
```
--------------------------------------------------------------------------------

Would you like to use one of the existing sets of values (1), or would you like
to enter a new set of values (2)?: 2
Please enter strictly positive values of capacitance, inductance, dissipation ra
te and charge percentage respectiveley:
100
100
100
0.4

The function exists on the interval [-2.000000, 2.000000]

Please enter start and end values for replotting the function and searching for
the root:
0
2

The root has been marked

The root (resistance) is R=0.896484 ohms and f(R)=0.000092

Would you like to save this set of values? (y/n): y
In which slot would you like to save the current set? Please enter a number from
 1 to 5: 3
```

Updated contents of the file after the user saves the data in slot 3:


C:\Users\mohamed\Desktop\FinalVersion\bin\Debug\FinalVersion.exe

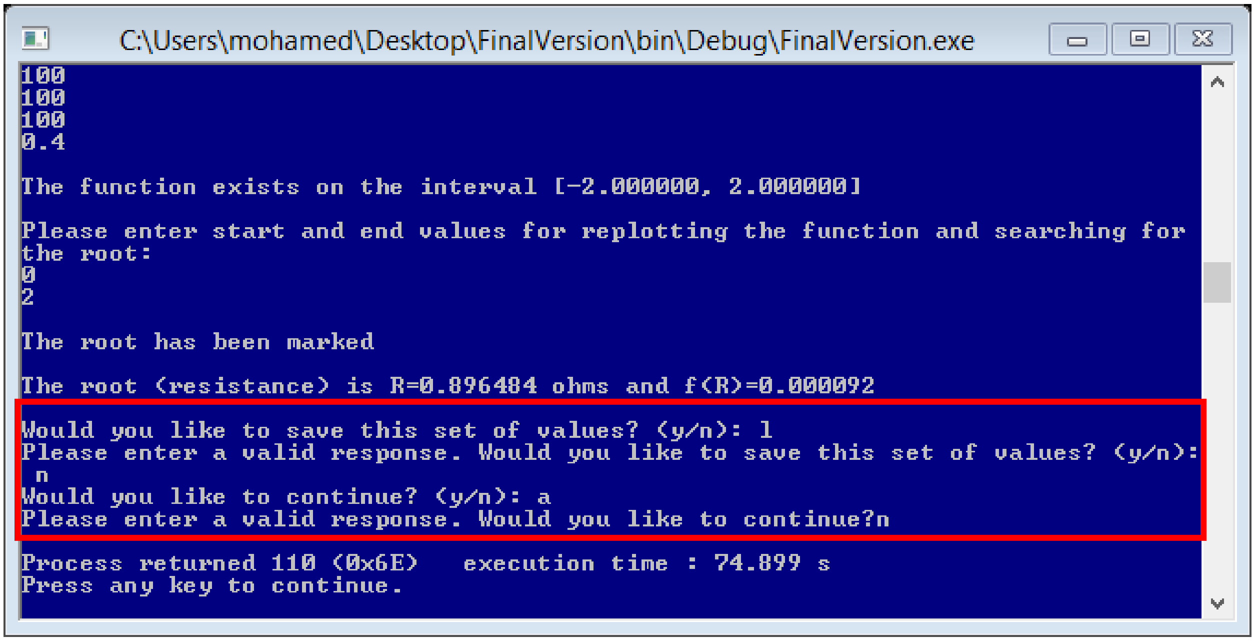
```
 1 to 5: 3
Would you like to continue? (y/n): y
File already exists. Here are the current sets of values in the file:

SET #1

Capacitance = 0.300
Inductance = 4.000
Dissipation rate = 0.500
Charge percentage = 0.700
--------------------------------------------------------------------------------
SET #2

Capacitance = 0.000
Inductance = 0.000
Dissipation rate = 0.000
Charge percentage = 0.000
--------------------------------------------------------------------------------
SET #3

Capacitance = 100.000
Inductance = 100.000
Dissipation rate = 100.000
Charge percentage = 0.400
--------------------------------------------------------------------------------
```

- Test case 13: Invalid choices from the user



C:\Users\mohamed\Desktop\FinalVersion\bin\Debug\FinalVersion.exe

```
Would you like to use one of the existing sets of values (1), or would you like
to enter a new set of values (2)?: 3
Please enter a valid response. Would you like to use one of the existing sets of
 values (1), or would you like to enter a new set of values (2)?: 1
Which set would you like to use? Please enter a number from 1 to 5: 1

The function exists on the interval [-7.302967, 7.302967]

Please enter start and end values for replotting the function and searching for
the root:
0
7

The root has been marked

The root (resistance) is R=4.713379 ohms and f(R)=0.000028

Would you like to continue? (y/n): h

Please enter a valid response. Would you like to continue?y

File already exists. Here are the current sets of values in the file:

SET #1
```



C:\Users\mohamed\Desktop\FinalVersion\bin\Debug\FinalVersion.exe

```
100
100
100
0.4

The function exists on the interval [-2.000000, 2.000000]

Please enter start and end values for replotting the function and searching for
the root:
0
2

The root has been marked

The root (resistance) is R=0.896484 ohms and f(R)=0.000092

Would you like to save this set of values? (y/n): l
Please enter a valid response. Would you like to save this set of values? (y/n):
 n
Would you like to continue? (y/n): a
Please enter a valid response. Would you like to continue?n

Process returned 110 (0x6E)   execution time : 74.899 s
Press any key to continue.
```