Course Project Documentation

CS101 Project

Equation solver

Group 386

Aman Virani, 140070009

Charvi Vitthal, 140070022

Supraja Tedla, 14D070059

Omkar Karmarkar, 140040009

Table of contents

1. Introduction :

It is very difficult to solve a complicated equation manually. So we decided to create an equation solver.

Even solving higher degree polynomials is a difficult task. But normal polynomials can be solved with the help of program by taking coefficients as input.

So, we decided to make an equation solver which can give roots of ‘ANY’ complicated equation/function which is continuous and differentiable for all real numbers.

The link to our demonstration video :-

https://www.youtube.com/watch?v=n0DMpam8ZKs

2. Problem statement :

Our aim was to build a program that can give the roots for given equation provided the function is continuous and differentiable over all real numbers.

One of the aim is to take input from user and parse it so that the equation can be solved by a numerical method.

To find the roots we were required to use new methods of finding roots. So our second aim was to find new numerical methods which are efficient and useful.

It is important that user is able to use the program easily. So our next aim was to create a nice and user friendly GUI.

3. Requirements :

Since we are creating a GUI, the system requires an application called ‘Qt creator’.

As the whole code is written in C++, a C++ compiler is required in order to run the code.

4. Implementation :

a)TOKENIZING:

Input is taken by user and ‘spaces’ are removed. If the user has used wrong brackets like “[“, “{“ , etc… These are replaced by plain brackets. If the user has given equation in the form of f(x)=g(x) , it is converted into f(x)-g(x)=0.f(x)-g(x) is taken as string. Then the string is divided into tokens by adding blank spaces in between the characters . A vector string is made by the characters of the input string . Variable is replaced by ‘x’ if user has input equation in any other variable or used ‘X’ instead of ‘x’.

b)SYNTAX CHECKER:

A syntax checker has been made to check whether the input equation is given in correct format.

Error message is shown on screen if one of these wrong syntax is found .

1) It is checked whether the variable (x) is present or not

2) Invalid tokens are present like any other variable or any not identified operator.

3) Unbalanced brackets

Ex:- (x+(x^2-(x+3))

4) mismatched brackets

Ex:- (x-4)*(x^3+45(

5) Invalid tokens around brackets

Ex:- (x-5)*(x-5+)

6) Invalid tokens around operators

Ex:- x+5*+4

7) Invalid tokens around variables and numbers

Ex:- x+4)-6

c)REVERSE POLISH NOTATION

One way to parse the given expression is to convert it into some intermediate notation that the computer can understand. we chose the reverse polish notation because of its simplicity in understanding and using. To convert our regular infix notation into the reverse polish notation we used a version of the shunting yard algorithm.

The shunting yard algorithm uses 2 stacks initially. one to store the operators and the other to store the constants and the variables. finally the RPN form is stored into a vector.

Then we created an evaluate program which simplifies the given RPN by plugging in the values of the variables present if any.

d)SOLVING FOR ROOTS

The basic algorithm we've used to obtain the roots is the newton raphson method. The biggest hurdle in applying this method was finding an initial approximation that converges to the root.

We took another approach to this problem. We first took the mid point of the interval first specified, and checked whether it is a root. If it is not, then we check the value of the derivative at that point. If the derivative is nonzero, then we applied newton raphson method to the point until it converges. When it converges to a root, we recursively apply the same function to the left limit to the root, and also to the root to the right limit. If the derivative at that point is zero, we recursively apply the same function to the left limit to that mid point, and to the root to the right limit.

5. Testing Strategy and Data

Here are a few test cases that worked, and a few that didn't.

1. x-2=0
   Output 2.
2. (x-1)*(x-2)=0
   Output 2, 1
3. cosh(x)-3=0
   Output 1.7627, -1.7627
4. (x-)=0
   Syntax Error
5. sin=0
   Syntax Error

The following test cases didn't work

1. (x-1)*(x-2)*(x-3)*(x-4)*(x-5)*(x-6)=0
   Output 6, 1
2. sin(x)=0
   When this function was used and the domain entered was the default domain, only roots from -100 to 100 were displayed, with a few roots missing.

6. Discussion of System:

A. What are worked as per plan?

1. Parsing the input

The tokenized input string has to be converted in a form which can be easily understood and used in the algorithm . So we have used RPN (Reverse Polish Notation) to convert the tokenized string into a vector of string which can be converted to RPN form by the Shunting Yard algorithm.

2. Newton Raphson Method

The newton raphson method works as promised, although the algorithm used for getting the roots fails for some cases, like (x-1)*(x-2)*(x-3)*(x-4)*(x-5)*(x-6)=0. Even after a lot of debugging, we were not able to weed out all the bugs and make the program run perfectly. Around 25-30 hours were spent in debugging this part alone.

3. The GUI

We redesigned the GUI into a more succinct design, so that the user can input the equation and get the answer in one window itself.

B. What we added more than discussed in SRS?

1. The UI was redesigned so that it is simpler for the user.

C. Changes made in plan:

1. Expression Tree

Expression Tree was initially part of our parsing function but later we found that we can effectively parse the given string without Expression Tree by using RPN

2. Change in Algorithm

We have changed the core algorithm which was not efficient in finding all the roots. The old algorithm was, plotting all the points (at regular intervals) in the specified domain and storing all the points at which the function is between a specified interval close to y=0. Newton Raphson method was to be applied at all those points to get the roots.

A new algorithm is made which efficiently gives the roots in a specified domain, which was discussed earlier

3. Change in GUI

The main window and output windows specified as two different windows but we have made a single window which has the both of them in same window.

7. Future Work:

A. In our current project we solve equations/functions which are continuous and differentiable . By finding the point discontinuity and dividing the domain into two parts and applying the method which we used earlier roots can be found . Therefore the project can be extended to solve discontinuous and non differential equations also.

B. A graph plotter can be attached to the project. A graph plotter can be made by using koolplot library .

C. We have solved equations in one variable , a project which can solve multivariable equations could be made by modifying the core core algorithm.

8. Conclusions:

Making this project was a very valuable learning experience for us. Although the program in the current state doesn't work perfectly, we feel it can be fixed with knowledge of higher level mathematics. Even so, it greatly enhanced our knowledge of mathematical/computational concepts like root finding and numerical methods. Such a program in its fully working condition can be a powerful tool in the hands of mathematicians, physicists and chemists, to test their theories and get quick roots of functions. But that doesn't mean it's useless for other classes of people. Students can use this for quick reference, and future CS101 students can build on it to make more powerful stuff.

9. References:

1) Qt creator :

   http://doc.**qt.io/**

2) Reverse polish notation

   http://mathworld.wolfram.com/ReversePolishNotation.html

3) Shunting yard algorithm

   http://rosettacode.org/wiki/Parsing/Shunting-yard_algorithm

   http://www.youtube.com/watch?v=QzVVjboyb0s

4) Newton Raphson method

   http://en.wikipedia.org/wiki/Newton's_method

https://www.math.ubc.ca/~anstee/math104/104newtonmethod.pdf

and several other online sources.