Numerical Analysis Tool

CSPC-101

Paras Khosla

21103107

NIT Jalandhar

Table of Contents

1. Problem Statement
2. Novelty of the Idea
3. Numerical Differentiation
4. Numerical Integration
   - Riemann Sum
   - Simpsons' Rule
5. Root Approximation
   - Newton-Raphson Method
   - Bisection Method
   - Secant Method
6. Calculator History - File Handling
7. Limitations
8. Scope

Problem Statement

There are various problems in mathematics which can only be solved upto a certain level of precision or accuracy because of the **non-existence** of **closed-form expressions** in many cases, which is why, the need for numerical methods arises. Various numerical methods' algorithms have been laid out by mathematicians and programmers which serve to fulfill this purpose. Under this project, I have programmed these algorithms into C language code, for the tasks of **differentiation, integration and root approximation** using various approaches as well as methods.

Novelty of the Idea

None of the popular open-source mathematical engines offer **freedom** in choosing the numerical method the user wishes to employ to solve the problem. Instead, they use various integration strategies in the back-end.

The novelty of this project is that the user gets to choose **the numerical method of their choice**, desired accuracy and the number of iterations of the algorithm.

Numerical Differentiation

$$\boxed{\left.\frac{df}{dx}\right|_{x=x_0} = \lim_{\epsilon\to 0} \frac{f(x_0+\epsilon)-f(x_0-\epsilon)}{2\epsilon}}$$

Numerical Integration

- Riemann Sum
    - Left Riemann Sum
    - Right Riemann Sum
    - Midpoint Rule
    - Trapezoidal Rule
- Simpsons' Rule

Numerical Integration - Riemann Sum

- **Left Riemann Sum**
  `leftIntegral(f,a,b,n)`. It computes the value of the input function at each left endpoint of the partition.
- **Right Riemann Sum**
  `rightIntegral(f,a,b,n)`. It computes the value of the input function at each right endpoint of the partition.
- **Midpoint Rule**
  `middleIntegral(f,a,b,n)`. It computes the value of the input function at the midpoint of each partition.
- **Trapezoidal Rule**
  `trapezoidal(f,a,b,n)`. It computes the areas of the trapezoids that the area under the function gets divided into.

Numerical Integration - Simpsons' Rule

$$\boxed{\int_a^b f(x)dx = \frac{\Delta x}{3}\left(f(x_0) + 4f(x_1) + 2f(x_2) + \cdots + 4f(x_{n-1}) + f(x_n)\right)}$$

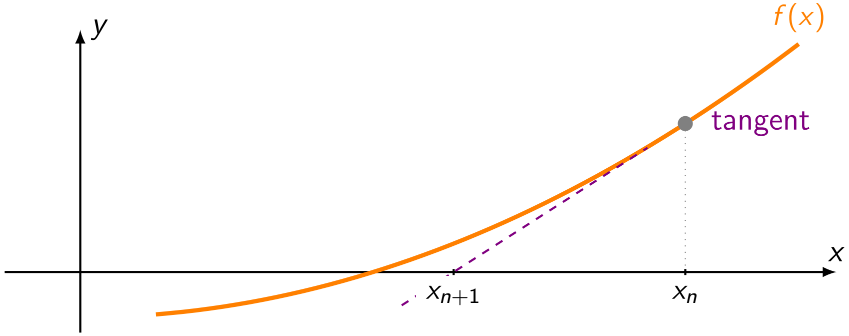
This computation can be done in a single stroke, instead of dividing the formula definition into two separate cases, as the pattern might otherwise suggest. We can simply deal with `oddTermSum` and `evenTermSum`, and then ultimately recombine the sum.

Root Approximation

- Newton-Raphson Method
    - `for` - `while` loop approach
    - Recursive approach
- Bisection Method
- Secant Method

Root Approximation - Newton-Raphson Method

$$\boxed{x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}}$$



Newton-Raphson Method - for-while approach

Defined by `printNewtonRaphson(f,eps,x0,maxIterations)`. Two parameters exist for deciding the termination of this loop based approach.

Depending on whether the desired accuracy specified by eps is reached first, which corresponds to the condition $|f(x_0) - 0| < \epsilon$ or the number of specified `maxIterations` has been performed, the function terminates, giving the approximate root obtained for the equation $f(x) = 0$, along with the computation at each iteration in a tabular format.

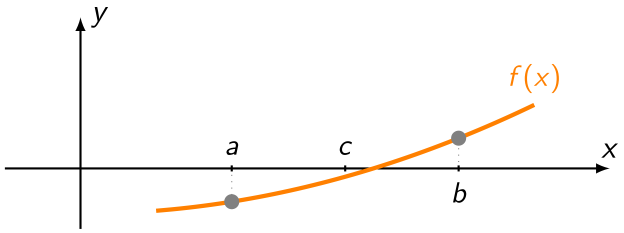
Newton-Raphson Method - Recursive approach

We can break the problem at hand into a smaller one by assigning the `double secondLastValue` as the result of the recursive function call on `newtonRaphsonRecursion()`, with input parameter `maxIterations-1`.

For termination, we can again apply, both the conditions, that of accuracy, and that of the maximum number of iterations allowed.

For all non-terminating conditions, we compute the "next guess", or the so called `lastValue` from the "previous guess" or the `secondLastValue`, and finally get to the approximate root of the equation $f(x) = 0$.

Root Approximation - Bisection Method



For continuous functions, if for any two points $a, b \in \mathbb{R}$, $f(a) \cdot (b) < 0$, that implies there must exist at least one root in (a, b). This method exploits this fact, and checks which value the root is more closer to by further checking this inequality between each one of the endpoints and the midpoint of (a, b), which happens to be at $c = \frac{a+b}{2}$. Again, this method is employed to that interval which contains the solution until either the desired accuracy is reached, or the specified number of iterations have not been performed.

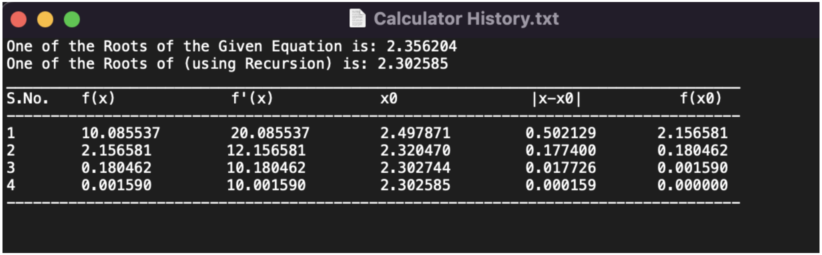
Root Approximation - Secant Method

$$\boxed{x_n = \frac{x_{n-2} f(x_{n-1}) - x_{n-1} f(x_{n-2})}{f(x_{n-1}) - f(x_{n-2})}}$$

Defined by `printSecant(f,x1,x2,eps,maxIterations)`. It takes the guesses `x1` and `x2` as input from the user, then keeps on improving on the guess by "drawing secants" between the points corresponding to these initial guesses, and keeps iterating until the desired accuracy is reached, or the specified number of iterations have not been performed, that is it terminates on fulfilment of any one of these conditions.

Calculator History - File Handling

The function `fprintNewtonRaphson(f,eps,x,maxIterations)` writes the iterations and results of the Newton-Raphson method using both recursion and `for-while` loop method into the file `Calculator History.txt`, using concepts of File Handling.



```
One of the Roots of the Given Equation is: 2.356204
One of the Roots of (using Recursion) is: 2.302585

_________________________________________________________________________________
S.No.    f(x)             f'(x)            x0               |x-x0|           f(x0)
---------------------------------------------------------------------------------
1        10.085537         20.085537       2.497871         0.502129         2.156581
2        2.156581         12.156581        2.320470         0.177400         0.180462
3        0.180462         10.180462        2.302744         0.017726         0.001590
4        0.001590         10.001590        2.302585         0.000159         0.000000
---------------------------------------------------------------------------------
```

Limitations

- **Function Input**: Challenging to take the function, which is being called on by other functions, as an input from the user directly as an expression.
- **Numerical Error Term**: The function `derivative(f,x0)` has been called numerous times in various other functions. This adds to the error term, since the derivative returned by this function is itself erroneous numerically, although of however small order it may be.

Scope

- **LaTeX Compilation**: The project can in the future be modified to add functionality to present the results as a compiled or typeset LaTeX document to add to the sophistication of the presentation.
- **Additional Mathematical Functions**: More mathematical operations can be carried with suitable algorithms, built on suitable data structures to cut down on time complexity.
- **Educational Gaming**: Certain games can be built along the lines of these code snippets. For example, in situations where an exact solution of an equation can be obtained, we might take input from the user to make an initial guess, and then compute the solution after a specified number of iterations of a different algorithm. If the estimated answer is within some predetermined error of the real solution, we may declare the guess *good* and assign a point to the user.