INDIAN INSTITUTE OF TECHNOLOGY,
BOMBAY

CS101 PROJECT

NumCpp:Project Report

Sandesh Kalantre
Anuj Shetty
Sanjana Sabu

Project Description:
Numerical library for various routines such as integration,differentiation,linear algebra, Fast Fourier Transform(FFT),etc. **with arbitary precision support**

Contents

1 Introduction

Numerical routines have various applications in scientific computing as well as in data analysis.C++ being a low-level language with explicit memory management allows us to have a better control over our data-structures.This is turn allows highly optimised code for computing various numerical routines.Hence the NumCpp project in C++ will allow the user to calculate various operations such as integration, differentiation, etc numerically optimally.

The project will allow the user to do the following operations:

- define a variable
- define a function with single as well as multiple arguments
- integrate single variable functions over a closed interval
- find the derivative of a single variable function at a given point
- define matrices and do operations such as inverse, transpose, etc
- solve simultaneous equations
- find the root of a function given an initial guess
- calculate the FFT for a discrete array of complex numbers

2 Functional Requirements

The program will be written and tested on a *nix environment.Hence it should work flawlessly on a *nix environment provided the following requirements are also met.

- *nix OS
- g++ 4.8.2
- make
- C++11(may be required depending on the final implementation of data structures)
- GNU Multiple Precision Arithmetic Library(GMP)[1]
- GNU MPFR Library[2]

[1]GMP `https://gmplib.org/`
[2]MPFR `http://www.mpfr.org/`

3 Structure of the Program

3.1 Parser

The user as mentioned above will have the ability to define functions using the keyword *define*. The functions will be given in infix notation [3]. However this form is quite difficut to evaluate.Hence it is converted to a more managable form known as the postfix notation or Revere Polish Notation(RPN) [4]. This conversion can be efficiently done by **Dijkstra's Shunting Yard algorithm**.[5].

This conversion of infix to RPN will be implemented in the class Parser.A class Function will be created to store function attributes such as arguments, the parsed RPN form of the function expression to allow faster evaluation, etc. Functions and variables will be stored in C++ STL container unordered map [6] of the class Function and doubles respectively.Unordered map is being used because it allows faster access than an ordinary map to the elements given the key.

3.2 Parser implementation

3.2.1 class Token

The basic objective of the token class is to generate a token which then can be used by the parser.A token is basically the smallest element of the expression which can be said to have an independent meaning like a number, variable name, operators like +,- *, etc.

The important members of the Token class are:

- **token_type** :stores the type of token such as a variable, function, number, etc.
- **token** :stores the token as <string> from STL.
- **get_token** :*function* which identifies the token given the expression to parse and the iterator to the first element of the expression. It returns the iterator to the next element after the token is parsed.

3.2.2 class Parser

This class contains the most important function **math_parse**.The mathematical expression consists of a set of tokens of numbers,variables and functions.However they are in infix form which is difficult and *slow* to evaluate for a computer.Hence they are converted by the math_parse function into Reverse Polish Notation which is extremely easy to evaluate. The expression once in RPN form is stored in an std::queue[7] of Tokens where each token has all the atributes stored as described in the Token class.

[3]Infix Notation `http://www.en.wikipedia.org/wiki/Infix_notation`
[4]Reverse Polish Notation `http://www.en.wikipedia.org/wiki/Reverse_Polish_notation`
[5]Shunting Yard Algorithm `http://en.wikipedia.org/wiki/Shunting-yard_algorithm`
[6]unordered map `www.cplusplus.com/reference/unordered_map/unordered_map/`
[7]Queue `www.cplusplus.com/reference/queue/queue/`

3.2.3 map_variables

The variables are stored in an unordered_map<std::string var_name, double> which then can be accessed when required.The map also stores values of important constants such as pi and e.

3.2.4 class Function

This class is used to define obejcts for user defined functions.This class stores the function name and number of arguments. The class has a method evaluate which when supplied with a vector<double> of arguments of the size of number of arguments returns the value of the function at that point.

3.2.5 map_functions

This is similar to the map_variables.The functions are stored in an unordered_map<std::string function_name,Function>. The functions then can be accessed by the map and the method evaluate in the class Function.

3.3 Differentiation

Differentiation effectively refers to the rate at which the value of a function changes. We will be differentiating the function at a given point x by evaluating the function at various points around x.

3.3.1 Differentiation using the definition

$$f^{'}(x) = \frac{f(x+h) - f(x)}{h}$$

For smaller values of x, this definition becomes more accurate. This actually is calculating the slope of the line joining two points $(x, f(x))$ and $(x+h, f(x+h))$. As we know, slope is calculated using the formula

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

3.3.2 Differentiation by taking average of slopes on either side of the given point

Using the same formula of slope, we will calculate the slope to the right of x i.e. slope of the line joining the points: $(x, f(x))$ and $(x + h, f(x + h))$. Similarly we will calculate the slope to the left of x i.e. the slope of the line joining the points $(x - h, f(x - h))$ and $(x, f(x))$. We will take the average of these two slopes to be $f'(x)$.

3.4 Integration

In this program, we will be using various numerical methods of integration to solve given definite integrals, and using the fact that the integral of a function between 2 endpoints represents the area bounded by the function and the x-axis.

3.4.1 Riemann integration

We divide the region between the end points of the integral (from a to b), into n equal parts of width h with h being very small for a good approximation of the integral by these rectangles of width h. Using this concept, there are some variations to the method available.

Method 1 We make the height of each rectangle of width h equal to the function value at the midpoint of each interval, to generally minimize error compared to if we take function values at only left endpoints of each interval or only right endpoints of each interval. Then we sum the area of these n rectangles, that is, sum the product of h and the function value of the midpoint of each interval.

Method 2 We make the height of each rectangle the function value of a random point in each interval, thus creating a tagged partition. Then we sum the area of each of these rectangles to give the integral.

Method 3 This method has unequal intervals. We check the magnitude of the slope at the midpoint of each interval, and if it is too large we will decrease the value of h, thus making narrower rectangles and increasing the accuracy of the approximation, and if it is too low, we will increase the value of h to make it more efficient by not wasting iterations.

Note:- When we speak of summing the areas of the rectangles, the area of a rectangle can be negative, if the function value we are considering in that interval is negative.

3.4.2 Monte Carlo integration

We take a large number of random points in the rectangle with the base with the same endpoints as that of the integral, and such a height so that the rectangle contains the entire function in that region. We then find the ratio of points which are inside the area bounded between the curve and the x-axis to the total points taken. This ratio is equal to the ratio of the area bounded by the function and the x-axis to the total area of the large rectangle chosen which contains the funcion between a and b.

3.4.3 Romberg method

This is obtained by repeatedly extrapolating the trapezium rule to better approximate the integral. We define:

$$h_n = \frac{(b-a)}{2^n}$$

$$R(0,0) = \frac{(b-a)(f(a)+f(b))}{2}$$

$$R(n,0) = \frac{R(n-1,0)}{2} + h_n \sum_{k=1}^{2^{(n-1)}} f(a+(2k+1)h_n)$$

$$R(n,m) = \frac{(4^m R(n,m-1) - R(n-1,m-1))}{(4^m - 1)}$$

The error for $R(n,m)$ from the value of the integral is of the order $(h_n)^{(2m+2)}$.

3.5 Root Finder

The objective of the root finder is to find a root x where the given contiuous function $f(x)$ cuts the x-axis on the graph, that is, $f(x) = 0$. This can be done in multiple ways, using different parameters, such as -

3.5.1 Newton-Raphson's algorithm

1. The root finder takes $f(x)$ and an initial guess of the root, say x_0, from the user.
2. We make an improved second guess by finding the point $(x1, 0)$ on the x-axis which the tangent at $(x0, f(x_0))$ passes through.
3. We then iterate this process until we have an x_n which gives $f(x_i) = 0$, or indistinguishably close to 0.
4. Any general x_i is given by: $x_i = x_{(i-1)} - \frac{f(x_{i-1})}{f'(x_{i-1})}$
5. If we encounter a critical point, i.e. where $f'(x_i) = 0$, it implies that the tangent is parallel to the x-axis and will not intersect it, therefore we suitably make changes in the value of x_i.

3.5.2 Bisection method

1. We take 2 values a and b from the user, such that $f(a)$ and $f(b)$ are of opposite signs. Since $f(x)$ is continuous, this implies that the curve cuts the $x - axis$ between $x = a$ and $x = b$.
2. Therefore we calculate the function value at the midpoint i.e. $f(\frac{a+b}{2})$ and check its sign.
3. If it has the same sign as $f(a)$, this means it has the sign opposite to that of $f(b)$ and therefore we can repeat the above process between $\frac{a+b}{2}$ and b, in a domain of half the size.
4. Similarly if it has the same sign as $f(b)$, we repeat the process between $f(a)$ and $f(\frac{a+b}{2})$.
5. Thus we iteratively refine the range of possible values of x for the root, until we finally encounter the point where $f(x) = 0$.

3.6 Solving simultaneous equations

We shall solve simultaneous equations using by representing them in matrix format and then using Gaussian's elimination to convert it into an upper triangular matrix and then we shall solve for the variables. There are a few special cases too like when there are infinite solutions and no solution.

1. Convert the coefficient matrix into upper triangular form.
2. Make all diagonal elements 1.If this can't be done, then the system does not admit a unique solution
3. Solve for each of the variables.

3.7 Fast Fourier Transform

A fast Fourier transform (FFT) is an algorithm to compute the discrete Fourier transform (DFT) and its inverse. The Fourier transform is an important equation for spectral analysis, and is required frequently in engineering and scientific applications. The FFT is an algorithm for computing a DFT that operates in $Nlog_2(N)$ complexity versus the expected N^2 complexity of a naive implementation of a DFT. The FFT achieves such an impressive speed-up by removing redundant computations.The algorithm used for this purpose is known as the Cooley-Tukey algoritm.[8]
For the implementation of the algorithm, an array of complex numbers will be accepted and stored using the `complex`[9] class in C++ STL. However it seemed simpler just to define such a class in our program and hence that implementation was present in the final version of the program.

4 Data structures

The following data structures will be used in the program.They are STL containers and were used to avoid rewriting standard functions and also to optimize the code which is one of the main requirements of a numerical library.

- `vector`[10]
  `vector` is similar to an array except that now we can create an array of any given type including classes.But unlike arrays, their size can change dynamically, with their storage being handled automatically by the container. `vectors` are very efficient at accessing its elements (just like arrays) and relatively efficient at adding or removing elements from its end.
- `stack`[11]
  This container provides Last in First Out(LIFO) access.Elements are inserted and extracted only from one end of the container. `stacks` will be used in implementations of various algorithms in the program.
- `map`[12]
  `map` is used to access elements provided the key to it.This container will be used for storing user defined variables and functions and the elements of the map will be accessed by the key namely the name of the variable or the function.

[8]Cooley-Tukey `http://en.wikipedia.org/wiki/Cooley-Tukey$_$FFT$_$algorithm`
[9]complex `http://www.cplusplus.com/reference/complex/complex/`
[10]vector `www.cplusplus.com/reference/vector/vector/`
[11]stack `www.cplusplus.com/reference/stack/stack/`
[12]map `www.cplusplus.com/reference/map/map/`

- `queue`[13]
  `queue` provides First in First out access.It will be used again in implementations of the algorithms in the class Parser.

5 Timeline for the Project

- Week 0
  Selection of the project topic.Discussion of the implementation of the project and algorithms that will be used.
- Week 1
  Implementation of the parser.Writing and testing code for integration and differentiation routines.
- Week 2
  Implementation of the linear algebra routines such as simultaneous equation solver,matrix operations,etc and the Fast Fourier Transform(FFT) routine.
- Week 3
  Writing help and documentation for the project.Extensive testing and error handling for bad inputs would be written. (If required)The GUI for the project will be written.This step is not needed in principle as the parser will have been implemented and the program can be effectively run from the command line.
- Week 4 Adding specialized functions for evaluation of Gamma function, Bessel functions, etc.number theoretic functions such as Euler's Totient function $\phi(n)$, etc.

6 Further scope

The Numerical library we have written can it extended to accomodate more routines as that for ODE solving,PDE solving etc.The integration routine could have been easily extended to integrate in multiple dimensions and in different coordinate systems.The project could have been appended with a graph-plotter for visualisation of the defined functions. Another possible extension is accepting a multi-variable function to maximise or minimise, given multiple constraint equations using a method similar to *Lagrange multipliers.*Another idea was to create an arbitary precision integer class which would have member functions for arithmetic over arbitarily large integers.

[13]queue `www.cplusplus.com/reference/queue/queue/`