COMPUTER ALGEBRA SYSTEM

Summer project-2011

Programming Club

Science & Technology Council

IIT Kanpur

By:-

Akshay Kumar, 10060, CSE

Abhijit Sharang, 10007, CSE

Mridul Verma, 10415, CSE

Bhargav Bhatt, 10195, MTH

CAS

A **computer algebra system** (**CAS**) is a software program that facilitates symbolic mathematics. The core functionality of a CAS is manipulation of mathematical expressions in symbolic form.

History of CAS

With the advent of new technology during the 1950's ,rigorous scientific and mathematical computation began to be done frequently.Much of the computation involved symbolic manipulation,which could be done only manually.This was tedious and tiresome. Especially in areas like artificial intelligence,a need began to be felt for computer programs that catered to the demands of the scientific community.Subsequently,the first computer program for symbolic mathematics,called **Schoonschip**,was developed by **Martin Veltman** in 1963.

The real breakthrough came in 1964,when **Carl Engelman** developed **MATHLAB** in 1964 at MITRE within an artificial intelligence research environment.It remained very popular for years to come.

Today,computer algebra software has become indispensible in colleges,universities ,research laboratories and engineering fields. The most popular commercial systems are **Mathematica Octave** and **Maple**, which are commonly used by research mathematicians, scientists, and engineers. Freely available alternatives include **Sage,GNU** and **Maxima.**

Components of CAS

Most powerful feature of CAS is related to Symbolic Manipulations. Besides this, most CAS softwares also do the following:

- Substitution of symbols or numeric values for certain expressions
- Handling Trigonometric functions , exponentials, logarithms *etc.*
- Partial and total differentiation
- Definite integration including multidimensional integrals
- symbolic constrained and unconstrained global optimization
- Solution of linear equations
- Solution of some differential and partial differential equation
- Matrix operations including products, inverses, *etc.*
- Statistical computation
- Fourier transforms

Important Algorithms Used

- Shunting Yard Algorithm
- Monte-Carlo's Integration
- Nelder-Mead Method
- Runge-Katta Method
- Finite Differences Method
- Cooley-Tokey Algorithm
- Secant Method
- Bisection Method

Description of Algorithm:

Shunting Yard Algorithm

This algorithm can be attributed as the backbone of the whole software. It has been named so as it resembles the operation of shunting railway tracks. This algorithm was invented by Edsger Dijkstra.

Algorithm:

Give preference to all the operators as follows:

- Transcendental functions-5
- ^ - 4
- * - 3
- / - 3
- + - 2
- - - 2

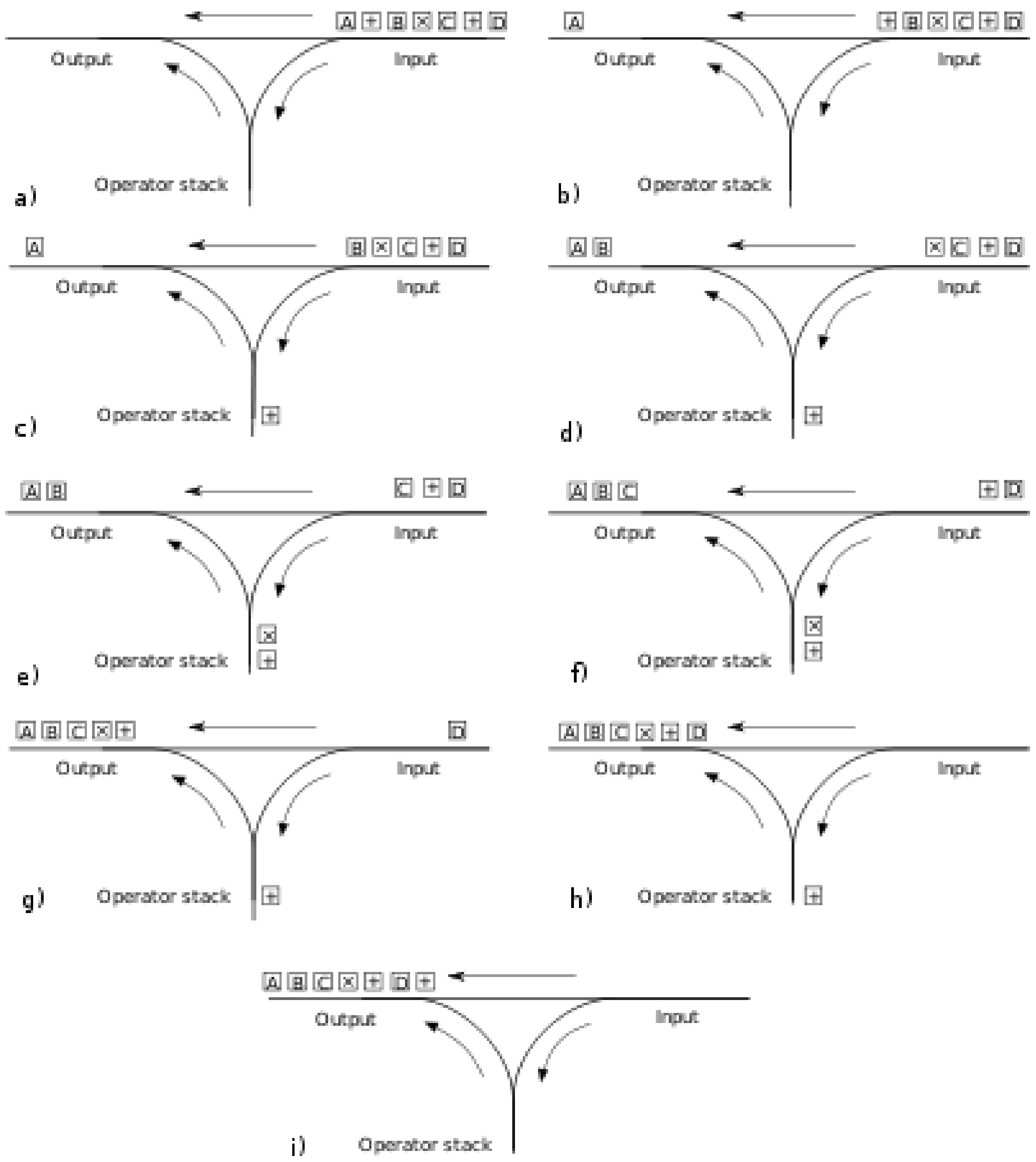
There are two arrays to be maintained: output queue and operator stack.

- While there are tokens to be read:
  - Read a token
  - If the token is a number, then add it to output queue.
  - If the token is a function token, then push it onto the stack.
  - If the token is a function argument separator (e.g., a comma):
    - Until the token at the top of the stack is a left parenthesis, pop operators off the stack onto the output queue. If no left parentheses are encountered, either the separator was misplaced or parentheses were mismatched.
  - If the token is an operator, o_1, then:
    - While there is an operator token, o_2, at the top of the stack, and either o_1 is left-associative and its precedence is less than or equal to that of $o_{2,\ o}$ r o_1 is right-associative and its precedence is less than that of o_2, pop o_2 off the stack, onto the output queue;
    - pop o_2 off the stack, onto the output queue;

- If the token is a left parenthesis, then push it onto the stack.
- If the token is a right parenthesis
  - Until the token at the top of the stack is a left parenthesis, pop operators off the stack onto the output queue.
  - Pop the left parenthesis from the stack, but not onto the output queue.
  - If the token at the top of the stack is a function token, pop it onto the output queue.
  - If the stack runs out without finding a left parenthesis, then there are mismatched parentheses.

- When there are no more tokens to read:
  - While there are still operator tokens in the stack;
  - If the operator token on the top of the stack is a parenthesis, then there are mismatched parentheses.
  - Pop the operator onto the output queue.
- Exit

To analyze the running time complexity of this algorithm, one has only to note that each token will be read once, each number, function, or operator will be printed once, and each function, operator, or parenthesis will be pushed onto the stack and popped off the stack once – therefore, there are at most a constant number of operations executed per token, and the running time is thus O(n) – linear in the size of the input.



Aforesaid diagram gives a pictorial representation on Shunting Yard Algorithm.

Monte-Carlo's Integration

In mathematics, Monte-Carlo Method is used for evaluating Multidimensional Integrals. The method consists of generating a hypercube with vertices at the maxima and minima of each dimension. For an integral I as follows,

$$I = \int_{a_1}^{b_1} dx_1 \int_{a_2}^{b_2} dx_2 \ldots \int_{a_n}^{b_n} dx_n \, f(x_1, x_2, \ldots, x_n) \equiv \int_V f(\bar{\mathbf{x}}) \, d\bar{\mathbf{x}}$$

The hypercube will have these dimensions:

$$\{\bar{\mathbf{x}}; a_1 \le x_1 \le b_1, a_2 \le x_2 \le b_2, \ldots, a_n \le x_n \le b_n\}$$

Next, pseudorandom numbers are generated within this region and the value of the function is computed at all these points. The integral is then estimated as:

$$I \approx Q_N \equiv V \frac{1}{N} \sum_{i=1}^{N} f(\bar{\mathbf{x}}_i) = V \langle f \rangle$$

Here, N is the number of points generated and V is the volume of the hypercube.

When the limits are dependent on other variables, the hyperspace is not a hypercube but a different region. For computing integrals in this region, we adopt the following strategy:

-Find the minima and maxima for each variable limit. This generates a hypercube

- Generate pseudorandom numbers within the hypercube as before
- Consider a function as follows:

  $g(x) = 1, if\ point\ lies\ in\ the\ hyperspace$

  $g(x) = 0, if\ the\ point\ lies\ outside\ the\ hyperspace$

- Integrate the function $f(x)g(x)$ within the hypercube using plane Monte Carlo Integration.

The sample variance of the integrand can be estimated using

$$\mathrm{Var}(f) \equiv \sigma_N^2 = \frac{1}{N-1}\sum_{i=1}^{N}(f(\bar{\mathbf{x}}_i) - \langle f \rangle)^2,$$

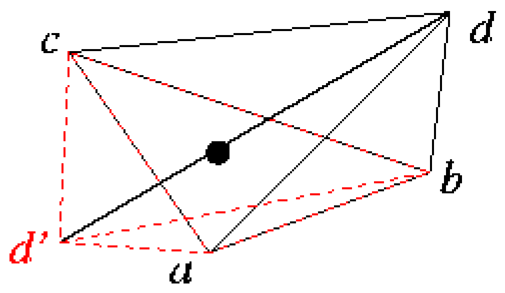
As long as the sequence $\{\sigma_1^2, \sigma_2^2, \sigma_3^2, \ldots\}$ is bounded, the variance decreases asymptotically to zero as 1/N.

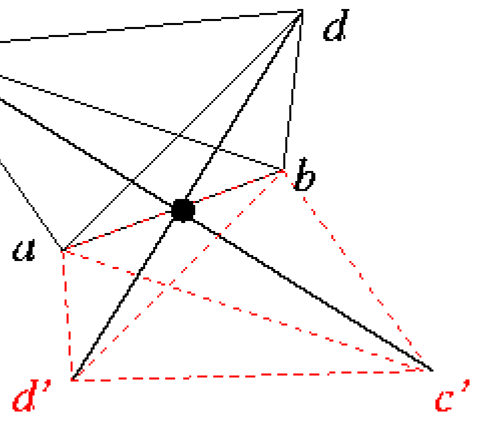
Nelder Mead Algorithm

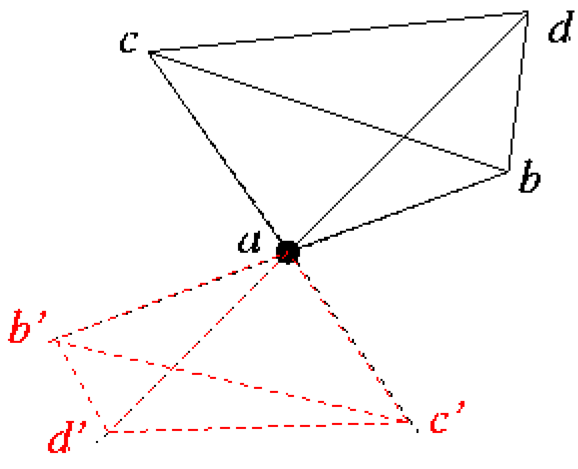
Nelder Mead is used to find out the minima and maxima of any type of functions but the result possibly depends on what you are using as a criterion of convergence i.e. whether the points are converging or the value of the function is converging. For discontinuous functions, values don't converge but the points converge. It can also be regarded as Downhill Simplex or Amoeba Method which uses non-linear optimization technique for convergence.

The points generated initially are assumed to form a degenerate simplex. If at any state the new point goes out of hypercube then it is started afresh.

In Nelder Mead Method, a simplex is reflected with respect to a point on its boundary. In the pictorial description on next page, the top figure shows the reflected simplex (drawn in red dashed line) when the point (the black dot) is the centroid of three vertices *a, b, c*. The middle figure shows the reflected simplex when the point is the centroid of two vertices *a, b*. The bottom figure shows the reflected simplex when the point is the vertex *a*.







Algorithm:

1. **Order** according to the values at the vertices:

$$f(\mathbf{x}_1) \leq f(\mathbf{x}_2) \leq \cdots \leq f(\mathbf{x}_{n+1})$$

2. Calculate x_o, the center of gravity of all points except x_{n+1}.

3. **Reflection**

   Compute reflected point $\mathbf{x}_r = \mathbf{x}_o + \alpha(\mathbf{x}_o - \mathbf{x}_{n+1})$

   If the reflected point is better than the second worst, but not better than the best, i.e.:$f(\mathbf{x}_1) \leq f(\mathbf{x}_r) < f(\mathbf{x}_n)$

   then obtain a new simplex by replacing the worst point x_{n+1} with the reflected point x_r, and go to step 1.

4. **Expansion**

   If the reflected point is the best point so far, $f(\mathbf{x}_r) < f(\mathbf{x}_1)$, then compute the expanded point $\mathbf{x}_e = \mathbf{x}_o + \gamma(\mathbf{x}_o - \mathbf{x}_{n+1})$. If the expanded point is better than the reflected point, $f(\mathbf{x}_e) < f(\mathbf{x}_r)$

   then obtain a new simplex by replacing the worst point x_{n+1} with the expanded point x_e, and go to step 1.

   Else obtain a new simplex by replacing the worst point x_{n+1} with the reflected point x_r, and go to step 1.

   Else (i.e. reflected point is not better than second worst) continue at step 5

5. **Contraction**

   Here it is certain that $f(\mathbf{x}_r) \geq f(\mathbf{x}_n)$

   Compute contracted product $\mathbf{x}_c = \mathbf{x}_{n+1} + \rho(\mathbf{x}_o - \mathbf{x}_{n+1})$

   If the contracted point is better than the worst point, $f(\mathbf{x}_c) < f(\mathbf{x}_{n+1})$

   then obtain a new simplex by replacing the worst point x_{n+1} with the contracted point x_c, and go to step 1.

   Else go to step 6.

6. **Reduction**

   For all but the best point, replace the point with

   $x_i = x_1 + \sigma(x_i - x_1)$ for all i $\in \{2, \ldots, n+1\}$. Go to step 1.

α, γ, ρ and σ are respectively the reflection, the expansion, the contraction and the shrink coefficient. Standard values are $\alpha = 1$, $\gamma = 2$, $\rho = 1 / 2$ and $\sigma = 1 / 2$.

For the **reflection**, since x_{n+1} is the vertex with the higher associated value among the vertices, we can expect to find a lower value at the reflection of x_{n+1} in the opposite face formed by all vertices point x_i except x_{n+1}.

For the **expansion**, if the reflection point x_r is the new minimum along the vertices we can expect to find interesting values along the direction from x_o to x_r.

Concerning the **contraction**: If $f(x_r) > f(x_n)$ we can expect that a better value will be inside the simplex formed by all the vertices x_i.

The initial simplex is important, indeed, a too small initial simplex can lead to a local search, and consequently the NM can get more easily stuck. So this simplex should depend on the nature of the problem.

Runge Kutta Method

Runge Kutta Method has been used as an explicit iterative method for soving ordinary differential equation. Runge Kutta Method can be thought as an extension of Euler's Method to fourth order thereby reducing the error term hugely. Fourth Order Runge Kutta Method is often reffered as "**RK4**"or "**classical Runge-Kutta method**".

Let initial value problem be specified as follows:

$$y' = f(t, y), \quad y(t_0) = y_0$$

The Runge Kutta method for this problem is given by following equations:

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$t_{n+1} = t_n + h$$

where y_{n+1} is the RK4 approximation of $y(t_{n+1})$, and

$$k_1 = hf(t_n, y_n)$$

$$k_2 = hf(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1)$$

$$k_3 = hf(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2)$$

$$k_4 = hf(t_n + h, y_n + k_3)$$

Thus, the next value (y_{n+1}) is determined by the present value (y_n) plus the product of the size of the interval (h) and an estimated slope. The slope is a weighted average of slopes:

- ✓ k_1 is the slope at the beginning of the interval;
- ✓ k_2 is the slope at the midpoint of the interval, using slope k_1 to determine the value of y at the point $t_n + h / 2$ using Euler's method;
- ✓ k_3 is again the slope at the midpoint, but now using the slope k_2 to determine the y-value;
- ✓ k_4 is the slope at the end of the interval, with its y-value determined using k_3.

In averaging the four slopes, greater weight is given to the slopes at the midpoint:

$$\text{slope} = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

The RK4 method is a fourth-order method, meaning that the error per step is on the order of h^5, while the total accumulated error has order h^4.

The method described above is applicable for first order Differential Equation but it can easily be extended to higher order Differential Equation. In these cases we consider a vector function of the given form:

$$\mathbf{y}(t) = (y(t), y'(t), y''(t), \ldots, y^{(N)}(t))$$

We use the above method to compute the (n-1)th derivative. The subsequent derivatives are computed iteratively until the base condition of computing y is arrived at. Since the method is fourth order, ordinary differential equation of upto 4th order can be solved without general loss of accuracy.

Cooley Tukey Method

The Cooley Tukey Method is used for computing Fast Fourier Transform (FFT) of a discrete series with reduces time complexity from $O(N^2)$ to O(n log N)_ It re-expresses the discrete Fourier transform (DFT) of an arbitrary composite size $N = N_1N_2$ in terms of smaller DFTs of sizes N_1 and N_2, recursively, in order to reduce the computation time to O(N log N) for highly-composite N (smooth numbers).

Algorithm

The Discrete Fourier Transform (DFT) is defined by the formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk},$$

Where k is an integer ranging from 0 to N-1

This method computes the DFTs of the even-indexed inputs x_{2m} ($x_0, x_2, \ldots, x_{N-2}$) and of the odd-indexed inputs x_{2m+1} ($x_1, x_3, \ldots, x_{N-1}$), and then combines those two results to produce the DFT of the whole sequence. This idea can then be performed recursively to reduce the overall runtime to O(*N* log *N*). This simplified form assumes that *N* is a power of two; since the number of sample points *N* can usually be chosen freely by the application, this is often not an important restriction.

This algorithm rearranges the DFT of the function x_n into two parts: a sum over the even-numbered indices $n = 2m$ and a sum over the odd-numbered indices $n = 2m + 1$

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N}(2m)k} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N}(2m+1)k}.$$

One can factor a common multiplier $e^{-\frac{2\pi i}{N}k}$ out of the second sum, as shown in the equation below. It is then clear that the two sums are the DFT of the even-indexed part x_{2m} and the DFT of odd-indexed part x_{2m+1} of the function x_n. Denote the DFT of the ***E***ven-indexed inputs x_{2m} by E_k and the DFT of the ***O***dd-indexed inputs x_{2m+1} by O_k and we obtain:

$$X_k = \underbrace{\sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2}mk}}_{\text{DFT of even-indexed part of } x_m} + e^{-\frac{2\pi i}{N}k} \underbrace{\sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2}mk}}_{\text{DFT of odd-indexed part of } x_m} = E_k + e^{-\frac{2\pi i}{N}k} O_k.$$

However, these smaller DFTs have a length of *N*/2, so we need compute only *N*/2 outputs: thanks to the periodicity properties of the DFT, the outputs for $N/2 \le k < N$ from a DFT of length *N*/2 are identical to the outputs for $0 \le k < N/2$. That is, $E_{k+N/2} = E_k$ and $O_{k+N/2} = O_k$. The phase factor exp[-2π*ik* / *N*] (called a twiddle factor) obeys the relation: exp[− 2π*i*(*k* + *N* / 2) / *N*]

$= e^{-\pi i}\exp[-2\pi ik/N] = -\exp[-2\pi ik/N]$, flipping the sign of the $O_{k+N/2}$ terms. Thus, the whole DFT can be calculated as follows:

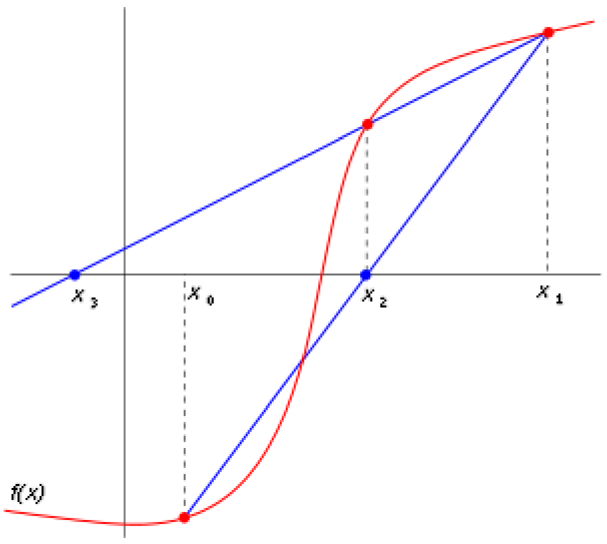
$$X_k = \begin{cases} E_k + e^{-\frac{2\pi i}{N}k} O_k & \text{if } k < N/2 \\ E_{k-N/2} - e^{-\frac{2\pi i}{N}(k-N/2)} O_{k-N/2} & \text{if } k \geq N/2. \end{cases}$$

Secant Method

Secant method is a root-finding algorithm that uses a succession of roots of secant lines to better approximate a root of a function *f*.

The Secant Method is defined by the Recurrence Relation:

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



The first two iterations of the secant method. The red curve shows the function *f* and the blue lines are the secants.

The Secant Method requires two initial values x_0 and x_1 which should ideally lie close to a root.

The iterates x_n of the secant method converge to a root of f, if the initial values x_0 and x_1 are sufficiently close to the root. The order of convergence is α, where

$$\alpha = \frac{1+\sqrt{5}}{2} \approx 1.618$$

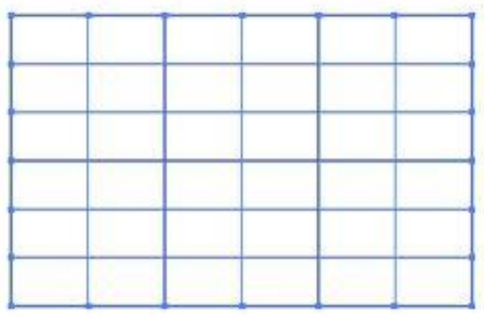
which incidentally comes out to be the "Golden Ratio".

Method of finite difference for partial differential equation

This method employs the approximation of a derivative as :

$$\frac{u(x+h)-u(x)}{h} \approx u'(x)$$

To solve a poisson's equation in two variables, we consider a rectangular region with boundaries as the x- axis, y- axis and two lines, one of which is parallel to the x-axis and the other is parallel to the y-axis. Next, the region is divided into square grids, each of which has an edge length h .Thus the region appears like this:



Let $x = ih$ and $y = jh$ for an arbitrary (x,y),where (i,j) is a grid coordinate and $2 \leq i \leq m-1$ and $2 \leq j \leq n-1$.

Now, the laplacian at (i,j) is approximated as :

$g_{i,j}=(u_{i+1,j}+u_{i-1,j}+u_{i,j+1}+u_{i,j-1}-4u_{i,j})/(h^2)$,where u is the unknown function.

This equation can be substituted by a matrix equation AU=B. Here A is:

$$A = \begin{bmatrix} D & -I & 0 & 0 & 0 & \dots & 0 \\ -I & D & -I & 0 & 0 & \dots & 0 \\ 0 & -I & D & -I & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -I & D & -I & 0 \\ 0 & \dots & \dots & 0 & -I & D & -I \\ 0 & \dots & \dots & \dots & 0 & -I & D \end{bmatrix}$$

I is the mxm identity matrix and D is an mxm matrix as follows:

$$D = \begin{bmatrix} 4 & -1 & 0 & 0 & 0 & \dots & 0 \\ -1 & 4 & -1 & 0 & 0 & \dots & 0 \\ 0 & -1 & 4 & -1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -1 & 4 & -1 & 0 \\ 0 & \dots & \dots & 0 & -1 & 4 & -1 \\ 0 & \dots & \dots & \dots & 0 & -1 & 4 \end{bmatrix}$$

u is the matrix: $[U] = [u_{11}, u_{21}, \dots, u_{m1}, u_{12}, u_{22}, \dots, u_{m2}, \dots, u_{mn}]^T$

B is the matrix obtained by computing values of g at every (x_i, y_j),considering the boundary conditions at *i*=2,*i*=m-1 and *j*=2,*j*=n-1.

It is evident that a point inside the rectangular region will have four grid points in its neighbourhood. We then consider the weighted mean of these points, the weights being dependent on the distance of the point under consideration from the aforesaid four points. Thus, we obtain the solution to the equation at a given point.

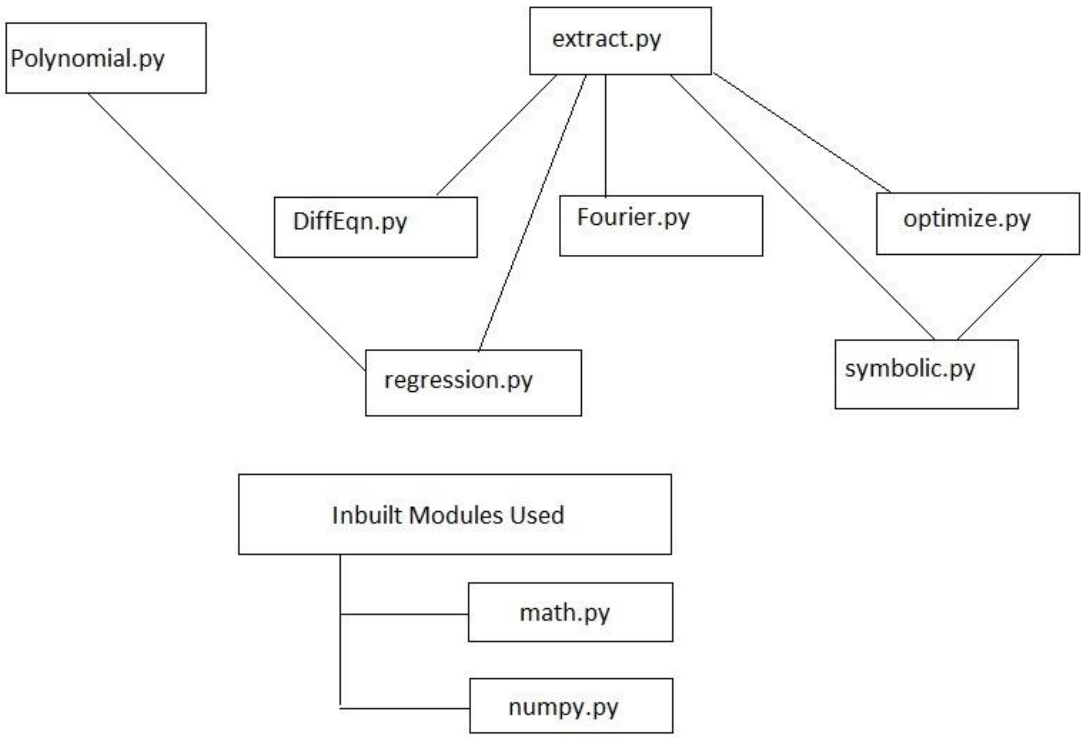
Bisection method

This is a simple and robust method for finding all the roots of an equation lying in an interval [a,b]. At each step the method divides the interval in two by computing the midpoint $c = (a+b) / 2$ of the interval and the value of the function $f(c)$ at that point. Unless c is itself a root (which is very unlikely, but possible) there are now two possibilities: either $f(a)$ and $f(c)$ have opposite signs and bracket a root, or $f(c)$ and $f(b)$ have opposite signs and bracket a root. The method selects the subinterval that is a bracket as a new interval to be used in the next step. In this way the interval that contains a zero of f is reduced in width by 50% at each step. The process is continued until the interval is sufficiently small.

Explicitly, if $f(a)$ and $f(c)$ are opposite signs, then the method sets c as the new value for b, and if $f(b)$ and $f(c)$ are opposite signs then the method sets c as the new a. (If $f(c)=0$ then c may be taken as the solution and the process stops.) In both cases, the new $f(a)$ and $f(b)$ have opposite signs, so the method is applicable to this smaller interval.

This method guarantees to converge to a root in an interval if there exists any. The absolute error is halved at each step so the method converges linearly, which is comparatively slow. Because of this, it is often used to obtain a rough approximation to a solution which is then used as a starting point for more rapidly converging methods.

Layout of our Software



As evident from the given flowchart, the function extract.py can primarily be attributed as the backbone of our software. It performs the task of calculating the value of a function on an n variable function ($n \leq 6$) at a particular point. The fact that $n \leq 6$ is not because of the limitation of the software but because of the fact that English allows only 26 alphabets out of which most of the alphabets find occurrence in transcendental functions which leaves only six alphabets desirable viz. x, y, z ,u, v, w (in order).

The module symbolic.py is basically concerned with the minutes of Calculus other than solving Differential Equations. The module symbolic.py can compute analytic as well as numeric differentiation of single dimensional or dimensional functions and also directional derivative. It can also handle numeric integration of single dimensional function and multi-dimensional functions. Other than this it can also compute analytic and numeric nth derivative of a single dimension function.

The module optimize.py performs the task of calculating the minima and maxima of a multi-dimensional function where the limits of a particular variable may be a function of other variables not only constant. This module is also called by symbolic.py while calculating numeric multidimensional integral.

Next in the pipeline is the module DiffEqn.py. As evident from its name, it is designed to compute the solution of explicit ordinary differential equation at a point. It can handle ordinary differential equation of a maximum of fourth order. As far as partial differential equations are concerned, it can solve Laplacian Equation in two dimensions. All the above functions of this module can only compute numeric solutions as solving a differential equation at some stage or the other encounters indefinite integration which this software is not acquainted with.

One module which doesn't rely on extract.py is polynomial.py. As its name suggests, it is designed to handle polynomial related computations. It can find out the sum, difference, product as well as quotient & remainder of two polynomials.

The module linked with polynomial.py is regression.py as it calls the object polynomial during computation of Legendre polynomials. The tasks performed by this module includes least square fitting, plane fitting, exponential fitting, logarithmic fitting & spline interpolation. The other statistical task performed by this module involves calculating Spearman's Coefficient, Pearson's Coefficient & Kendall Tau's Coefficient.

The module Fourier.py performs all the Fourier related tasks like calculating sin coefficient, cos coefficient & complex coefficient. In the field of Discrete Fourier Transformation, it can compute Fast Fourier Transform as well as Inverse Fourier Transform of a sequence.

Limitations of CAS:

- The software designed by us is incapable of computing indefinite integrations, an indispensible tool in present mathematics. As a result of this various facets of CAS could not be discovered for example solving symbolic differential equation.
- Those modules which give a symbolic answer do not have an inbuilt feature to simplify the expressions thereby giving a user readable answer. We tried improving upon this by simplifying the expressions as much as possible but since it doesn't involve a single general rule, full success could not be achieved.
- Both the above mentioned two points intensively uses heuristics which we are not acquainted with.
- Some algorithms deployed by us like Monte Carlo Integration, Nelder Mead Method, etc. are probabilistic in nature and hence do not give an exact answer. Moreover since Multidimensional Integration itself uses maximizing & minimizing a function the error term of both the function may add up.
- Since the GUI uses only two textboxes for input, the input must be given in a certain specified format as given in the help menu. The software doesn't accept input in any arbitrary fashion.
- It is more of a utility based software and should be used for routine mathematics manipulations. Testing the software for special cases or boundary values may lead to garbage output.

Future Potential of CAS

- As evident from its name, the more you dwelve into the subject the, more you realize how little you know. Hence this field is an always improving arena and many more functions can be added to be it.
- In future, we also intend to develop a text editor which can directly call the functions of our software give the desired output.
- The two major tasks not accomplished in this project, viz. indefinite integration and simplification of expression also needs to be looked upon as they present a greater challenge for us.

Frontend of the software:

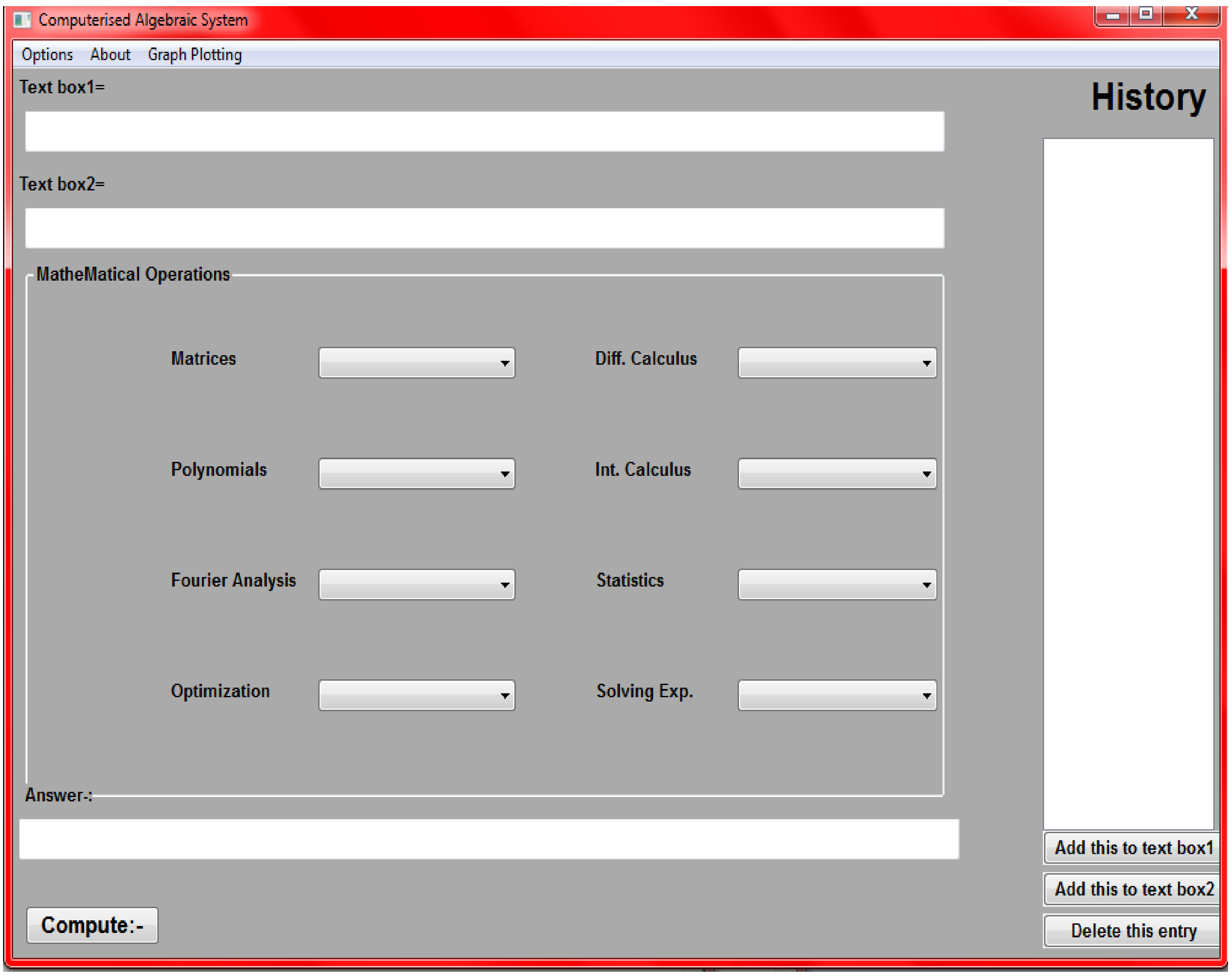
The Frontend of the software was made in wxpython . WxPython is the module of python which when included is used to make WxWidgets.

wxPython is a wrapper for the cross-platform GUI API (often referred to as a 'toolkit') wxWidgets (which is written in C++) for the Python programming language. It is one of the alternatives to Tkinter, which is bundled with Python. It is implemented as a Python extension module (native code). Other popular alternatives are PyGTK and PyQt. Like wxWidgets, wxPython is free software.

HISTORY OF WXPYTHON

wxPython was born when Robin Dunn needed a GUI to be deployed on HP-UX systems and also on Windows 3.1 in a few weeks time. While evaluating commercial solutions, he ran across Python bindings for the wxWidgets toolkit. Thus, he learned Python and, in a short time, became one of the main developers of wxPython (which grew from those initial bindings), together with Harri Pasanen. The first versions of the wrapper were created by hand. However, soon the code base became very difficult to maintain and keep in sync with wxWidgets releases. Later versions were created with SWIG, greatly decreasing the amount of work to update the wrapper. The first "modern" version was announced in 1998.

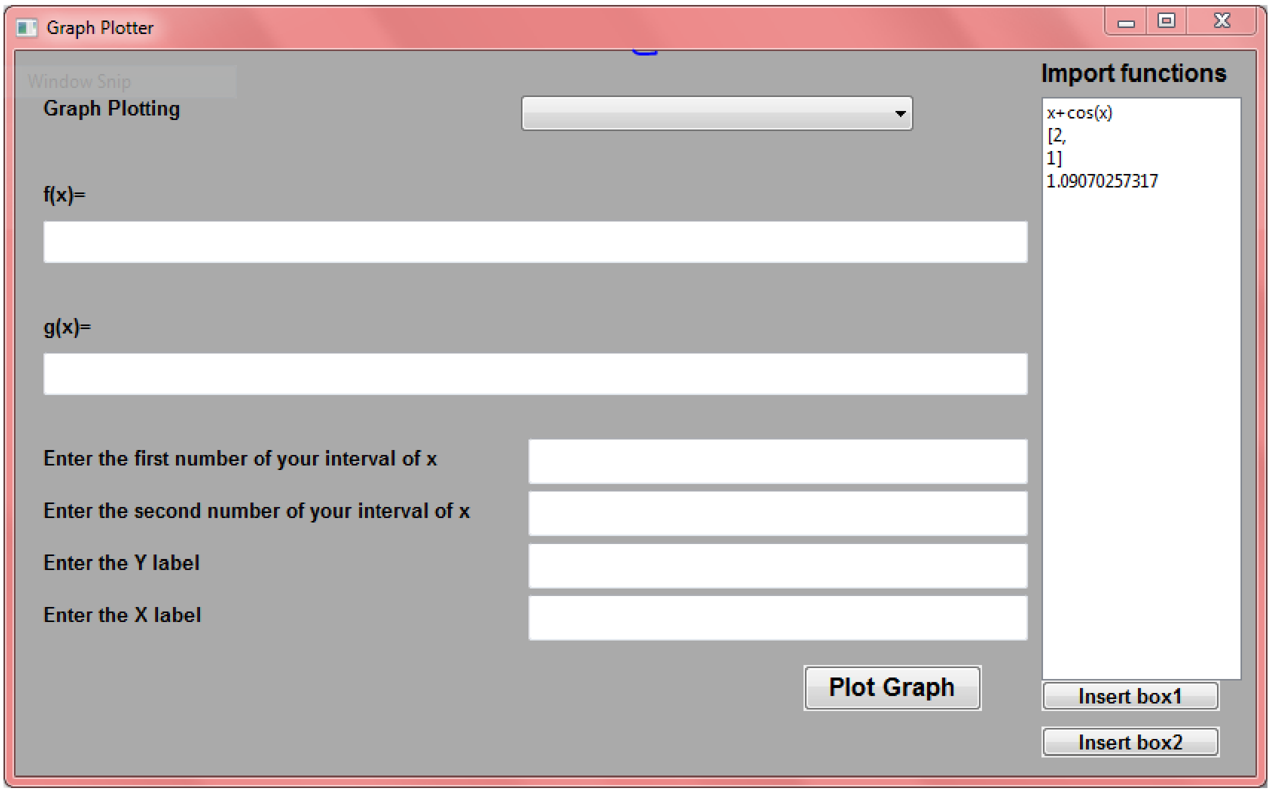
Snapshot:



The Graphical User Interface Includes a single window which contains two text boxes for entry of data and an answer text box to display the answer. The main feature of our GUI is that it can communicate easily with the user with the constraint that it takes the text in a particular format and If the text is given in some any other format is display an error.

Our GUI has some additional features as well like:

Graph Plotter



x+cos(x)
[2,
1]
1.09070257317

This window helps us in Plotting Graphs. It asks from the user various attributes for the graph.

The graph and main GUI both have a history menu from which they can call in the functions they require in the functions whenever they require .In this way they don’t have to write the whole function again and again.

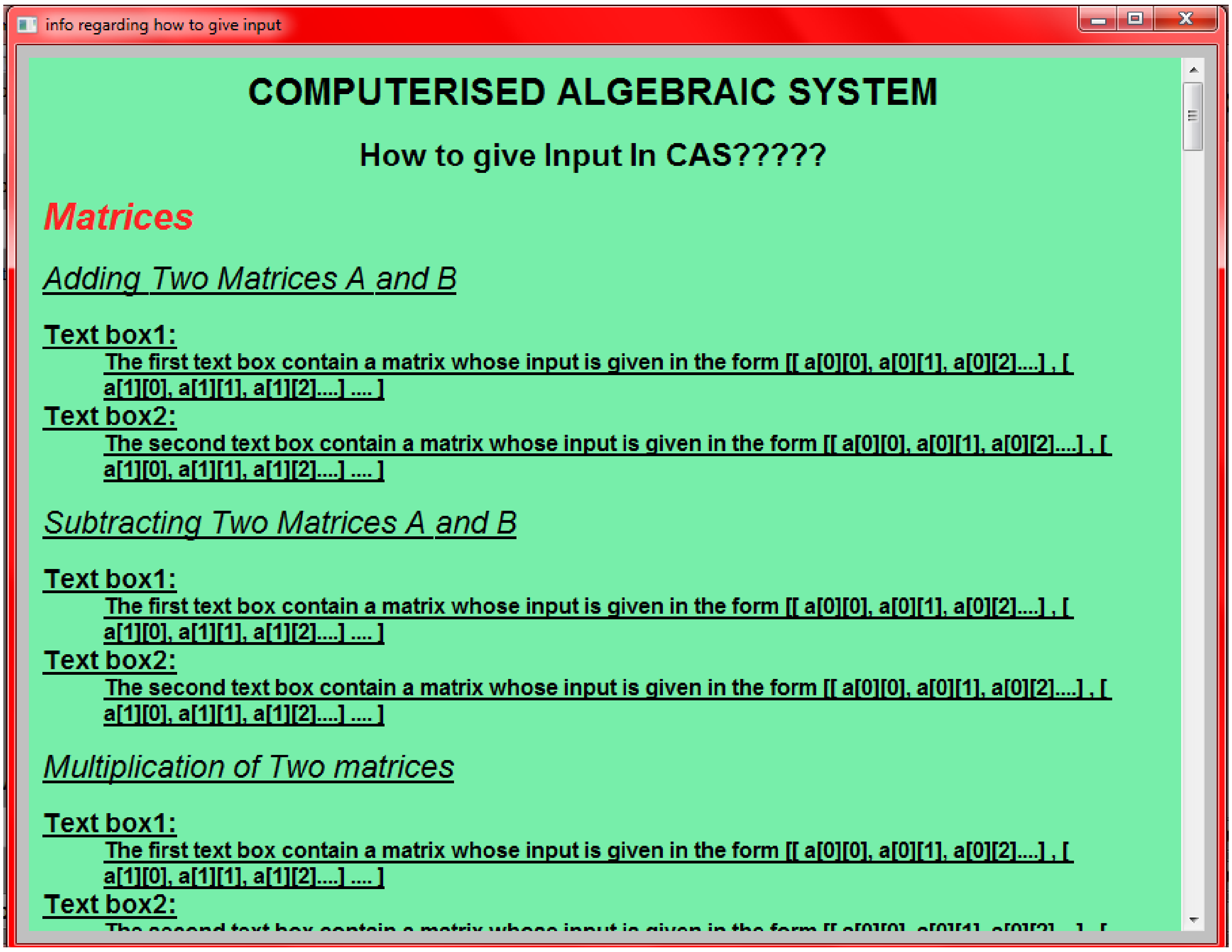
There are three types of graph plotting available in our GUI

- Plot f(x) vs x
- Plot f(x) vs g(x)
- Plot f(x) vs x and g(x) vs x

The whole GUI has been created keeping in mind that the user should feel comfortable while working in CAS.

The various other features of our GUI are :

- HELP : If you are not acquainted with the GUI you need to read it to become familiar to how to give input .



COMPUTERISED ALGEBRAIC SYSTEM

How to give Input In CAS?????

Matrices

Adding Two Matrices A and B

Text box1:
The first text box contain a matrix whose input is given in the form [[a[0][0], a[0][1], a[0][2]....] , [a[1][0], a[1][1], a[1][2]....] ....]

Text box2:
The second text box contain a matrix whose input is given in the form [[a[0][0], a[0][1], a[0][2]....] , [a[1][0], a[1][1], a[1][2]....] ....]

Subtracting Two Matrices A and B

Text box1:
The first text box contain a matrix whose input is given in the form [[a[0][0], a[0][1], a[0][2]....] , [a[1][0], a[1][1], a[1][2]....] ....]

Text box2:
The second text box contain a matrix whose input is given in the form [[a[0][0], a[0][1], a[0][2]....] , [a[1][0], a[1][1], a[1][2]....] ....]

Multiplication of Two matrices

Text box1:
The first text box contain a matrix whose input is given in the form [[a[0][0], a[0][1], a[0][2]....] , [a[1][0], a[1][1], a[1][2]....] ....]

Text box2:
The second text box contain a matrix whose input is given in the form [[a[0][0], a[0][1], a[0][2]....] , [

- Refresh :If you want all the text boxes and the ongoing calculations to be finished then use the refresh button in the menu.

- Clear: If you want to clear all the data in the history then use the clear in the menu bar.

- Clear an entry : If you want to delete a particular entry in the history then use the “Delete this entry “ just below the history .

References:
1.en.wikipedia.org
2.Learning Python- Mark Lutz, O’Reilly publication
3.jstor.org
4.ocw.mit.edu-Lecture notes by MIT
5.Google search engine