13/05/2016

Numerical Analysis Term Project

Ahmed Elsayed Mahmoud	05
Fady Yousry Adeeb	46
Mohamed Tarek Mohamed Saber	57
Mohamed Magdy Hosny Abdelsalam	62
Moatz Ahmed Youssef	66

PART ONE

PROBLEM STATEMENT

The aim of this assignment is to compare and analyze the behavior of the different numerical methods studied in class: Bisection, False-position, Fixed point, Newton-Raphson, Secant and Bierge Vieta.

You are required to implement a root finder program which takes as an input the equation, the technique to use and its required parameters (e.g. interval for the bisection method).

Also, you should implement a general algorithm that takes as an input the equation to solve and outputs its roots.

PSEUDOCODE

BISECTION METHOD

Pseudo code (Bisection Method)

1. Input $\in > 0$, $m > 0$, $x_1 > x_0$ so that $f(x_0)\ f(x_1) < 0$.

   Compute $f_0 = f(x_0)$.

   $k = 1$ (iteration count)

2. Do

   {

   (a) Compute $f_2 = f(x_2) = f\left(\dfrac{x_0 + x_1}{2}\right)$

   (b) If $f_2 f_0 < 0$, set $x_1 = x_2$ otherwise set $x_0 = f_2$ and $f_0 = f_2$.

   (c) Set $k = k+1$.

   }

3. While $|f_2| > \in$ and $k \le m$

   set $x = x_2$, the root.

FALSE POSITION

False Position Method (Pseudo Code)

1. Choose $\in > 0$ (tolerance on $|f(x)|$)
   m > 0 (maximum number of iterations)
   k = 1 (iteration count)
   x_0, x_1 (so that $f_0, f_1 < 0$)
2. {
   a. Compute $x_2 = x_1 - \left(\frac{x_1 - x_0}{f_1 - f_0}\right) f_1$
      $f_2 = f(x_2)$

   b. If $f_0 f_2 < 0$ set $x_1 = x_2$, $f_0 = f_2$

   c. k = k+1

   }
3. While ($|f_2| \geq \in$) and ($k \leq m$)
4. $x = x_2$, the root.

NEWTON METHOD

Newton's Method - Pseudo code

1. Choose $\in > 0$ (function tolerance $|f(x)| < \in$)
   m > 0 (Maximum number of iterations)
   x_0 - initial approximation
   k - iteration count
   Compute $f(x_0)$
2. Do { $q = f'(x_0)$ (evaluate derivative at x_0)
   $x_1 = x_0 - f_0/q$
   $x_0 = x_1$
   $f_0 = f(x_0)$
   k = k+1
   }
3. While ($|f_0| \geq \in$) and ($k \leq m$)
4. $x = x_1$ the root.

SECANT METHOD

The secant Method (Pseudo Code)

1. Choose $\in > 0$ (function tolerance $|f(x)| \leq \in$)
   $m > 0$ (Maximum number of iterations)
   x_0 , x_1 (Two initial points near the root)
   $f_0 = f(x_0)$
   $f_1 = f(x_1)$
   $k = 1$ (iteration count)
2. Do {
   $$x_2 = x_1 - \left(\frac{x_1 - x_2}{f_1 - f_0}\right) f$$
   $x_0 = x_1$
   $f_0 = f_1$
   $x_1 = x_2$
   $f_1 = f(x_2)$
   $k = k+1$ }
3. While ($|f_1| \geq \in$) and ($m \leq k$)

FIXED POINT

```
FUNCTION Fixpt(x0, es, imax, iter, ea)
  xr = x0
  iter = 0
  DO
    xrold = xr
    xr = g(xrold)
    iter = iter + 1
    IF xr ≠ 0 THEN
      ea = |(xr − xrold) / xr| · 100
    END IF
    IF ea < es OR iter ≥ imax EXIT
  END DO
  Fixpt = xr
END Fixpt
```

BIRGE VIETA

```
function [Y,iterations,precesion,time] = bierge_vieta(myfunction,X0,accuracy,maxIterations)
    coeffs <= get_coefficients(myfunction)
    xi <= X0
    Y  <= X0
    iterations <= 0
    ae = accuracy
    begin time
    for j from 1 to maxIterations
        b <= coeff[1]
        c <= coeff[1]
        for i from 2 to length of coeffs
            b <= coeff[i] + b * xi
            if it not last iteration
                c <= b + c * xi
        xi <= xi - b/c
        Y <= add last(xi)
        iterations++
        precesion <= | (Y[end] - Y[end - 1]) /  Y[end] |
        if precesion[end] < ae
            break
    end time
```

GENERAL ALGORITHM DESCRIPTION

If the user doesn't choose a method to find the root, the program will solve using the default numerical root finder method which is **Bisection method.** It has been chosen because it always find a root (false-position but sometimes it converges very slowly) and the number of iterations required to attain an error can be calculated:

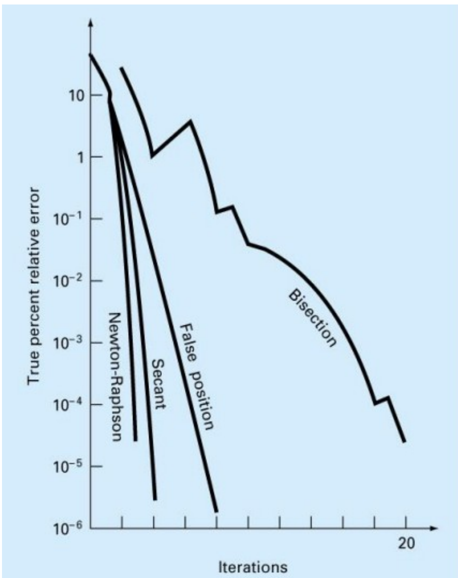
$$k \geq lg(|L_0/(x_1 * \varepsilon_{es})|)$$

DATA STRUCTURE

Arrays for storing different values after each iterations, precision and error

CONCLUSIONS

a) Newton Raphson is usually the fastest Method
b) Bisection and Fixed Point is usually the slowest method
c) Secant Method is faster than Regula Falsai
d) The Approximate Arrangement from the fastest to the slowest
    i. Newton Raphson
    ii. Secant
    iii. False Position (Regula Falsi)
    iv. Bisection
    v. Fixed Point



ANALYSIS AND PROBLEMATIC FUNCTIONS

BISECTION

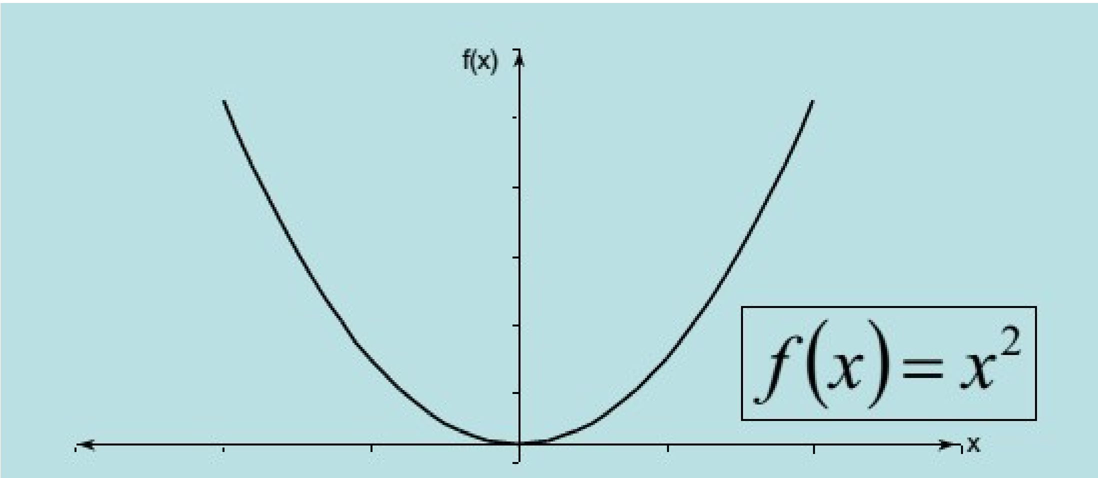
PROBLEMATIC FUNCTIONS

- $f(x) = x^2$

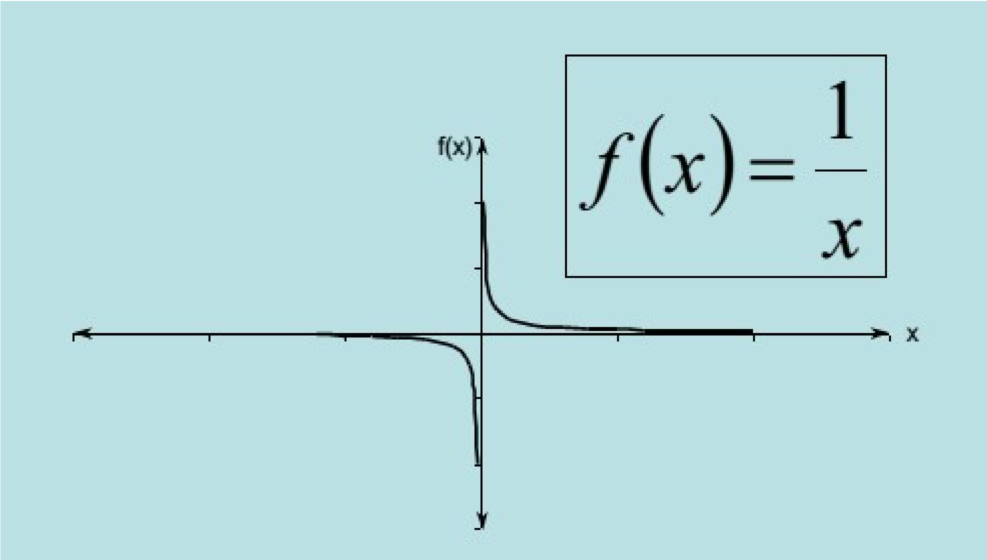
- $f(x) = 1/x$

ANALYSIS AND DRAWBACKS

- Slow
- Need 2 initial guesses
- If x is close to x_l and x_u this doesn't mean that f(x) is close to 0 and this can be solved by substituting by x_r and make sure that f(x) is close to 0
- If the function touches the x-axis we can't find two initial guesses (Even Multiple Roots)



- If the function changes sign but the root doesn't exist



FALSE-POSITION

PITFALLS

- Cannot detect continuous function.
- One of the two bounds can be stuck at one point and the convergence will be very slow so we can detect this case and use bisection to solve this problem or use maximum iterations to break this stuck
- If x is close to x_l and x_u this doesn't mean that f(x) is close to 0 and this can be solved by substituting by x_r and make sure that f(x) is close to 0
- Like bisection method initial guesses are required.

AVOIDING PITFALLS

One way to mitigate the "one-sided" nature of the false position is to have the algorithm detect when one of the bounds is stuck. If this occurs, then the original formula of bisection can be used.

DIVERGENCE AND CONVERGENCE

-If initial guesses is correct the method will converge but if initial guess is not correct it will diverge.

-number of iterations can not calculated before solving function

FIXED-POINT

ANALYSIS AND DRAWBACKS

- g(x) diverge or converge according to g'(x)

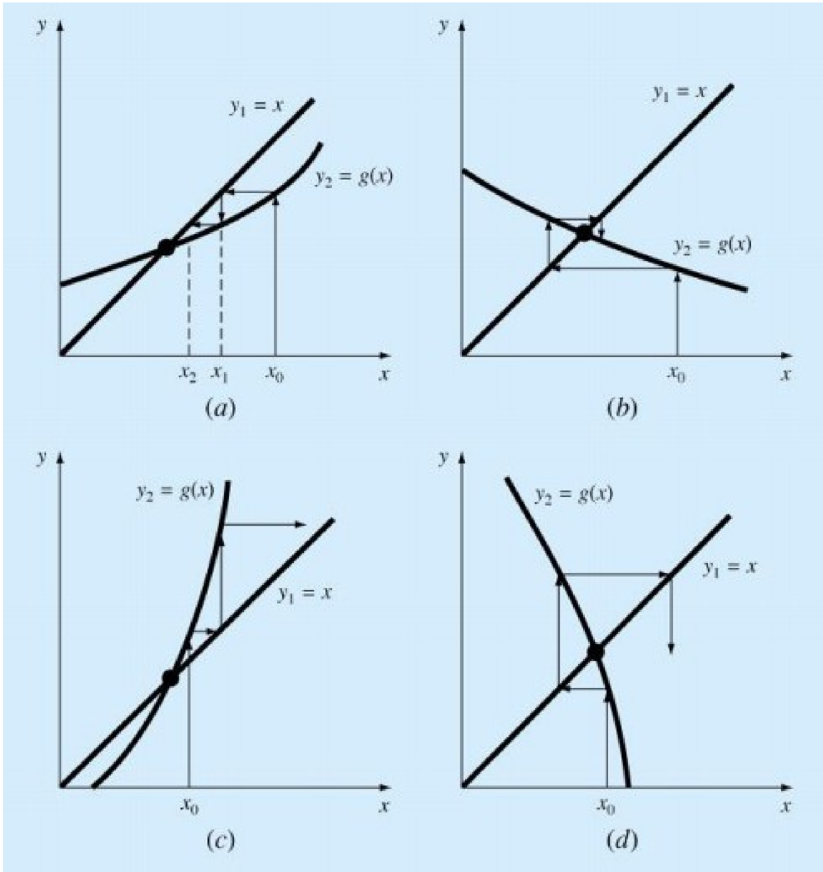
(a) $|g'(x)| < 1$, $g'(x)$ is +ve
$\Rightarrow$converge, monotonic

(b) $|g'(x)| < 1$, $g'(x)$ is -ve
$\Rightarrow$converge, oscillate

(c) $|g'(x)| > 1$, $g'(x)$ is +ve
$\Rightarrow$diverge, monotonic

(d) $|g'(x)| > 1$, $g'(x)$ is -ve
$\Rightarrow$diverge, oscillate

Demo



PITFALLS

-Finding the magical formula that will converge is the main pitfall.

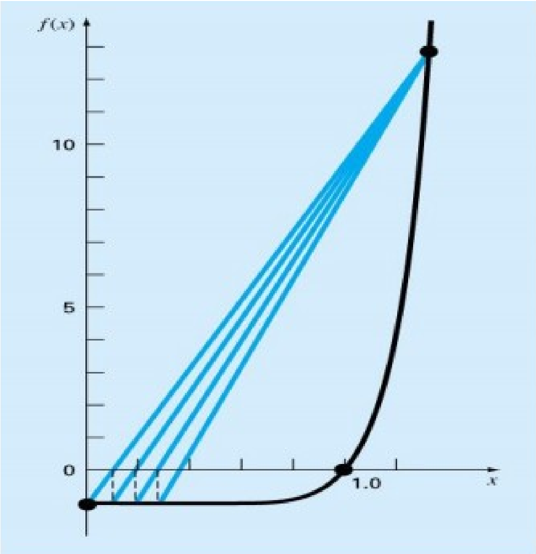
AVOIDING PITFALLS

Find first derivative and then substitute by the initial guess if less than |1| it will converge.

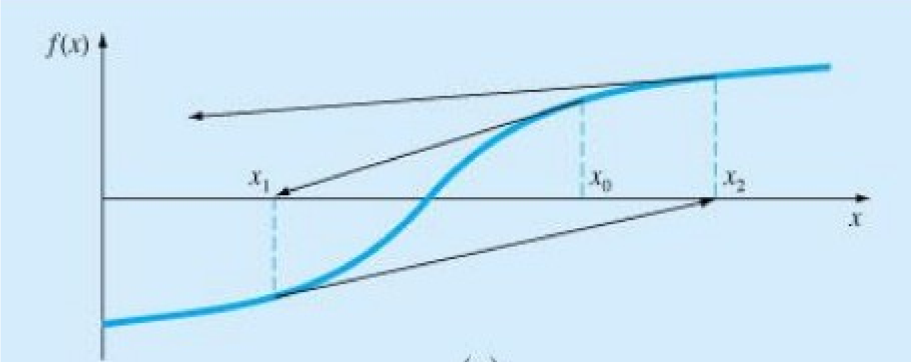
NEWTON

ANALYSIS AND DRAWBACKS

- sometimes slow (Multiple roots or stuck)

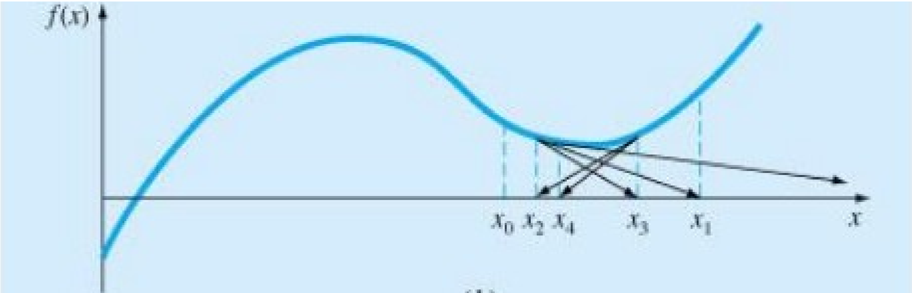


- It might diverge (Inflection point)



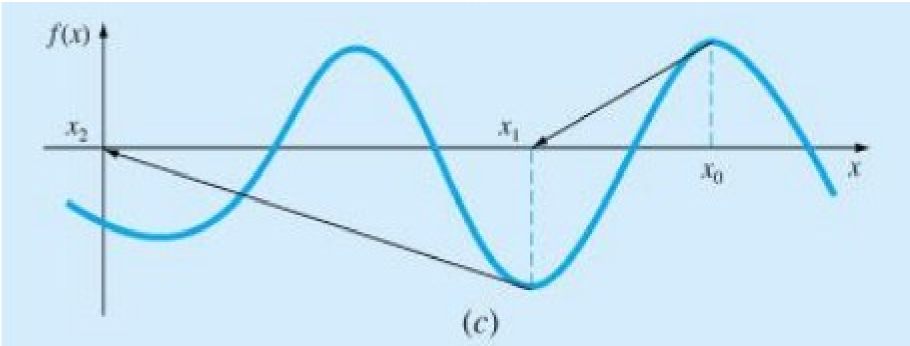
(a)

- local maximum or minimum

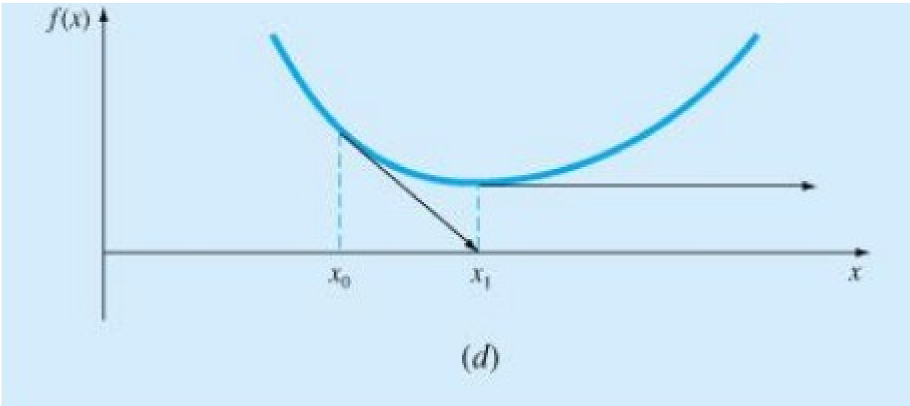


(b)

- it may jump from a place close to root to one that is far



- zero slope causes division by zero so we will break



SECANT

ANALYSIS AND DRAWBACKS

- It is slower than Newton Raphson
- Division by zero
- root jumping

DIVERGENCE AND CONVERGENCE

If it converged it will converge quadratic and there is no guarantee for convergence

BIRGE-VIETA

ANALYSIS AND DRAWBACKS

1. Find the root of x4 - 3x3 + 3x2 - 3x + 2 = 0

In this problem the coefficients are a0 = 2, a1 = -3, a2 = +3, a3 = -3, a4 = +1

Let the initial approximation to p be p0 = 0.5

a0 = 2.0	a1= -3.0	a2 = 3.0	a3 = -3.0	a4 = 1.0
b0 = 1.0	b1 = -2.5	b2 = 1.75	b3 = -2.125	b4 = 0.9375
c0 = 1.0	c1 = -2.0	c2 = 0.75	c3 = -1.75	

p1 = p0 - (b4 / c3) = 1.0357143

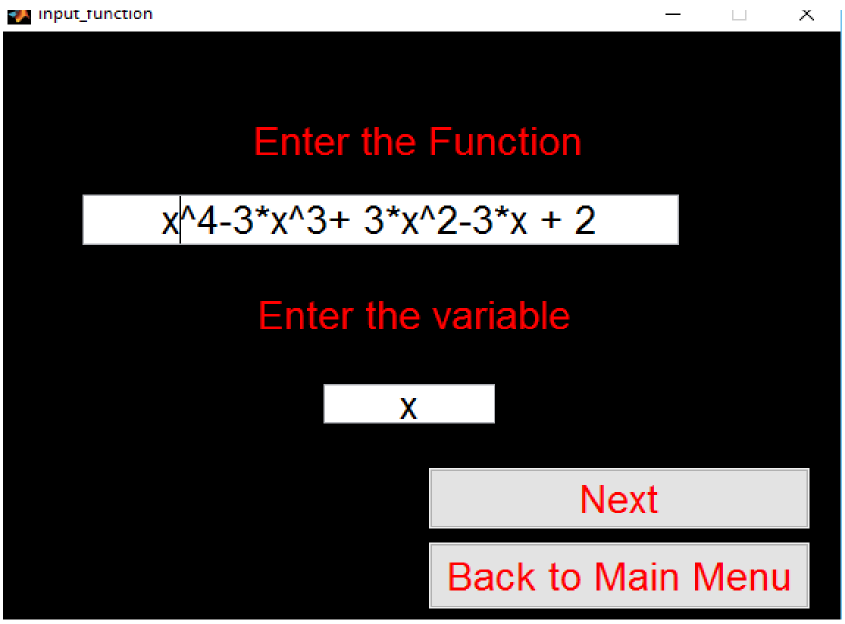
a0 = 2.0	a1 = -3.0	a2 = 3.0	a3 = -3.0	a4 = 1.0
b0 = 1.0	b1 = -1.964	b2 = 0.965	b3 = -1.999	b4 = -0.0714
c0 = 1.0	c1 = -0.928	c2 = 0.0038	c3 = -1.996	

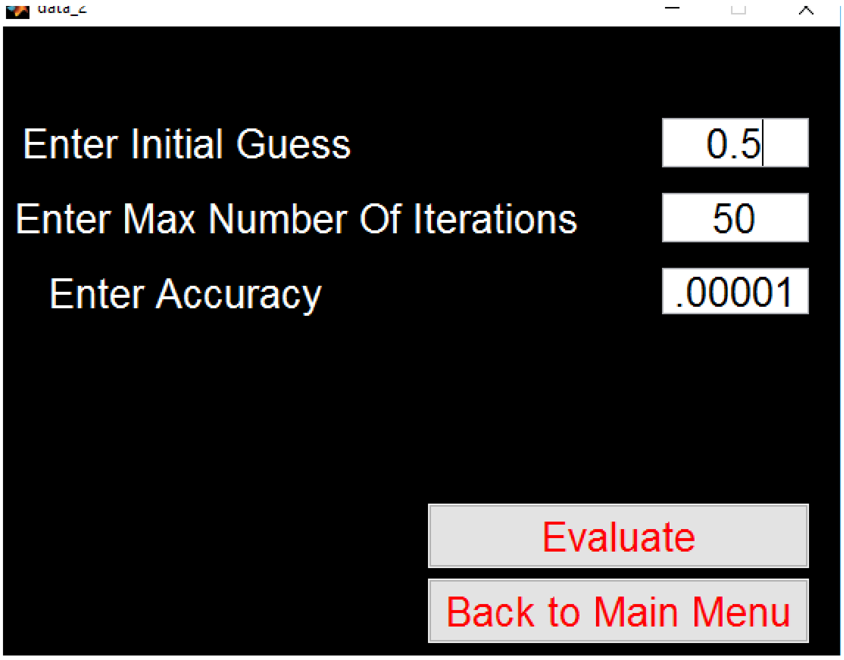
p2 = 0.9999518

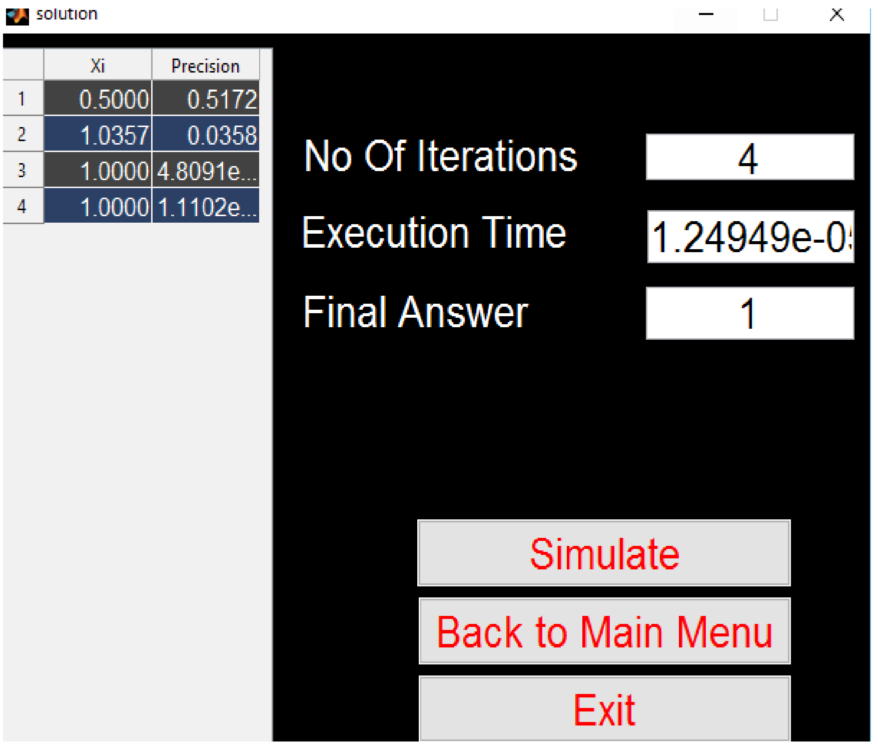
a0= 2.0	a1 = -3.0	a2 = 3.0	a3 = -3.0	a4 = 1.0
b0 = 1.0	b1 = -2.00	b2 = 1.00	b3 = -2.0	b4 = 9.64E-5
c0 = 1.0	c1 = -1.00	c2 = 2.38E-7	c3 = -1.999	

p3 = 1.0

So one of the root of the give equation is 1.0







2. Find the root of x4 - x - 10 = 0

In this problem the coefficients are a0 = 2, a1 = -3, a2 = +3, a3 = -3, a4 = +1

Let the initial approximation to p be p0 = 1.5

a0 = -10.0	a1 = -1.0	a2 = 0.0	a3 = 0.0	a4 = 1.0
b0 = 1.0	b1 = 1.5	b2 = 2.25	b3 = 2.375	b4 = -6.4375
c0 = 1.0	c1 = 3.0	c2 = 6.75	c3 = 12.5	

p1 = 2.0149999

a0 = -10.0	a1 = -1.0	a2 = 0.0	a3 = 0.0	a4 = 1.0
b0 = 1.0	b1 = 2.01	b2 = 4.06	b3 = 7.18	b4 = 4.47
c0 = 1.0	c1 = 4.029	c2 = 12.18	c3 = 31.72	

p2 = 1.87409

a0 = -10.0	a1 = -1.0	a2 = 0.0	a3 = 0.0	a4 = 1.0
b0 = 1.0	b1 = 1.87	b2 = 3.51	b3 = 5.58	b4 = 0.46
c0 = 1.0	c1 = 3.74	c2 = 10.54	c3 = 25.32	

p3 = 1.8558675

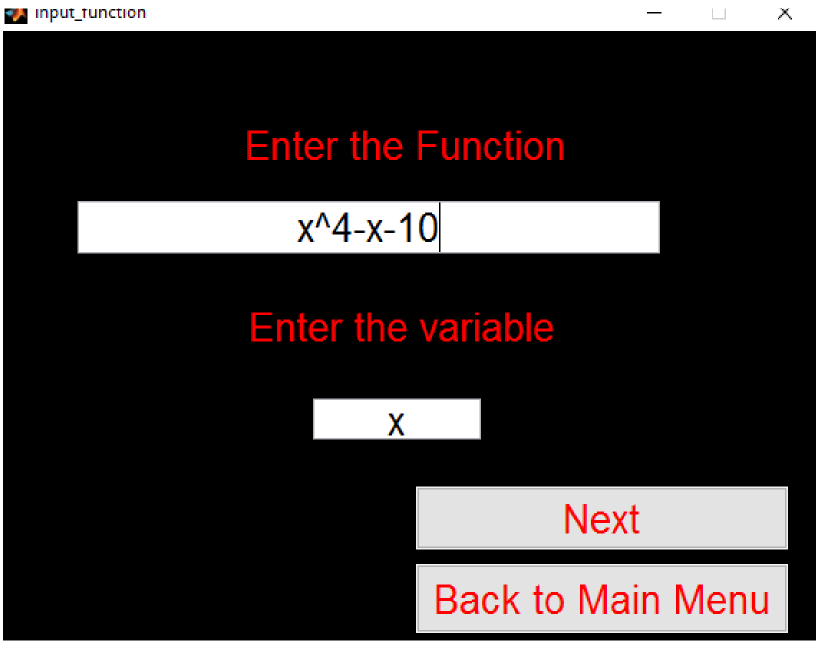
a0 = -10.0	a1 = -1.0	a2 = 0.0	a3 = 0.0	a4 = 1.0
b0 = 1.0	b1 = 1.85	b2 = 3.44	b3 = 5.39	b4 = 0.0069
c0 = 1.0	c1 = 3.71	c2 = 10.33	c3 = 24.56	

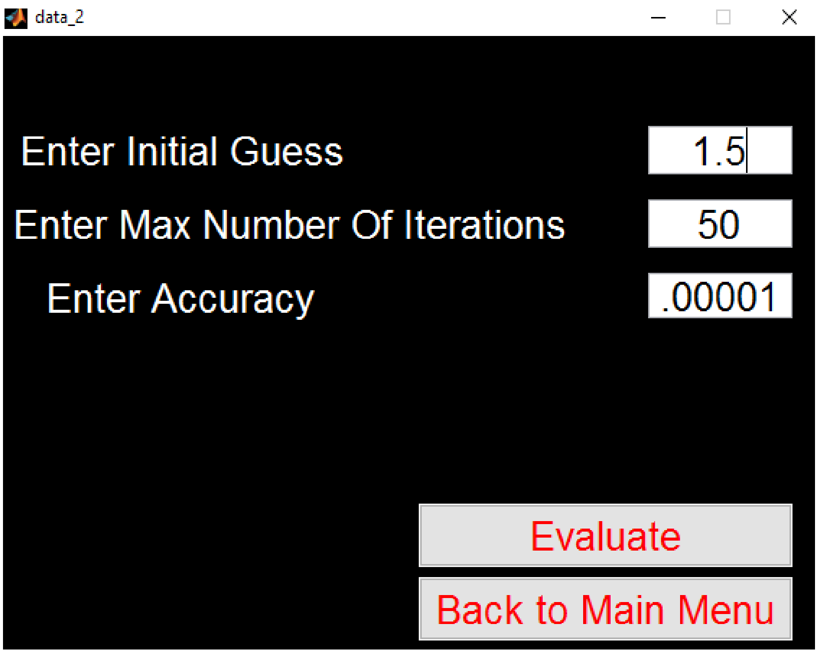
p4 = 1.8555846

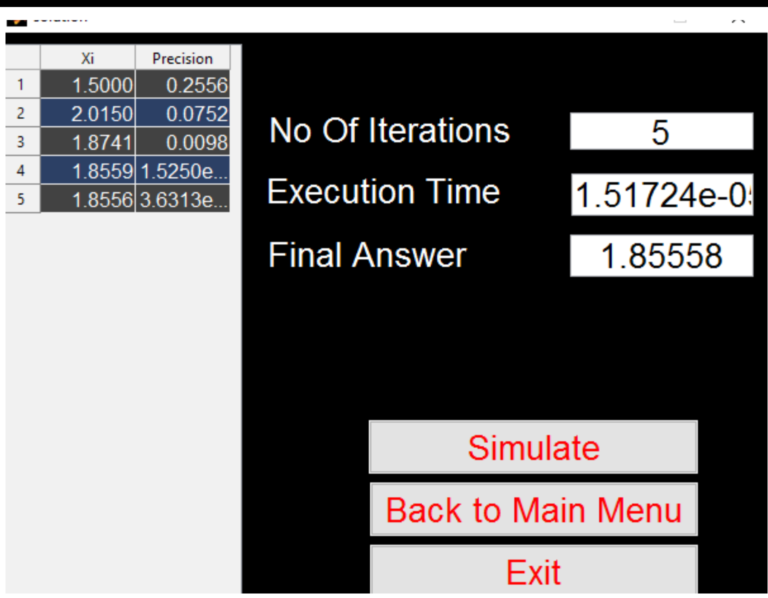
a0 = -10.0	a1 = -1.0	a2 = 0.0	a3 = 0.0	a4 = 1.0
b0 = 1.0	b1 = 1.855	b2 = 3.44	b3 = 5.38	b4 = 2.8E-6
c0 = 1.0	c1 = 3.71	c2 = 10.329	c3 = 24.556	

p5 = 1.8555845

so one of the root of the given equation is 1.8555845.


input_function
Enter the Function
x^4-x-10
Enter the variable
x
Next
Back to Main Menu


data_2
Enter Initial Guess
1.5
Enter Max Number Of Iterations
50
Enter Accuracy
.00001
Evaluate
Back to Main Menu


Xi
Precision
1 1.5000 0.2556
2 2.0150 0.0752
3 1.8741 0.0098
4 1.8559 1.5250e...
5 1.8556 3.6313e...
No Of Iterations
5
Execution Time
1.51724e-0
Final Answer
1.85558
Simulate
Back to Main Menu
Exit

- Vieta's formulas are formulas that relate the coefficients of a polynomial (**only polynomial**) to sums and products of its roots.

THEORITICAL BOUNDS :

1) BISECTION:

Length of the first Interval

$$L_0 = x_u - x_l$$

After 1 iteration

$$L_1 = L_0 / 2$$

After 2 iterations

$$L_2 = L_0 / 4$$

.....

.....

After k iterations

$$L_k = L_0 / 2k$$

Then we can write:

$$\varepsilon_a \leq \frac{L_k}{x} * 100$$

$$\varepsilon_a \leq \varepsilon_{es}$$

$$\left| \frac{L_k}{x_l} \right| \leq error\_tolerance$$

$$\left| \frac{L_0}{2^k} \right| \leq \left| x_l * \varepsilon_{es} \right|$$

$$2^k \geq \left| \frac{L_0}{x_l * \varepsilon_{es}} \right| \quad \Rightarrow \quad \boxed{k \geq \log_2 \left(\left| \frac{L_0}{x_l * \varepsilon_{es}} \right| \right)}$$

2) FIXED-POINT

According to the derivative mean-value theorem, if g (x) and g'(x) are continuous over an interval xi ≤ x ≤ α, there exists a value x = c within the interval such that

$$g'(c) = \frac{g(\alpha) - g(x_i)}{\alpha - x_i}$$

From (1) and (6), we have $\delta_i = \alpha - x_i$ and $\delta_{i+1} = g(\alpha) - g(x_i)$

Thus (7)

$$g'(c) = \frac{\delta_{i+1}}{\delta_i}$$

$$\delta_{i+1} = \delta_i \, g'(c)$$

- Therefore, if |g'(c)| < 1, the error decreases with each iteration. If |g'(c)| > 1, the error increase.
- If the derivative is positive, the iterative solution will be monotonic.
- If the derivative is negative, the errors will oscillate.

3) NEWTON-RAPHSON METHOD

6.2.1 Termination Criteria and Error Estimates

As with other root-location methods, Eq. (3.5) can be used as a termination criterion. In addition, however, the Taylor series derivation of the method (Box 6.2) provides theoretical insight regarding the rate of convergence as expressed by $E_{i+1} = O(E_i^2)$. Thus the error should be roughly proportional to the square of the previous error. In other words, the

Aside from the geometric derivation [Eqs. (6.5) and (6.6)], the Newton-Raphson method may also be developed from the Taylor series expansion. This alternative derivation is useful in that it also provides insight into the rate of convergence of the method.

Recall from Chap. 4 that the Taylor series expansion can be represented as

$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + \frac{f''(\xi)}{2!}(x_{i+1} - x_i)^2 \qquad \text{(B6.2.1)}$$

where ξ lies somewhere in the interval from x_i to x_{i+1}. An approximate version is obtainable by truncating the series after the first derivative term:

$$f(x_{i+1}) \cong f(x_i) + f'(x_i)(x_{i+1} - x_i)$$

At the intersection with the x axis, $f(x_{i+1})$ would be equal to zero, or

$$0 = f(x_i) + f'(x_i)(x_{i+1} - x_i) \qquad \text{(B6.2.2)}$$

which can be solved for

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

which is identical to Eq. (6.6). Thus, we have derived the Newton-Raphson formula using a Taylor series.

Aside from the derivation, the Taylor series can also be used to estimate the error of the formula. This can be done by realizing that if the complete Taylor series were employed, an exact result would be obtained. For this situation $x_{i+1} = x_r$, where x is the true value of the root. Substituting this value along with $f(x_r) = 0$ into Eq. (B6.2.1) yields

$$0 = f(x_i) + f'(x_i)(x_r - x_i) + \frac{f''(\xi)}{2!}(x_r - x_i)^2 \qquad \text{(B6.2.3)}$$

Equation (B6.2.2) can be subtracted from Eq. (B6.2.3) to give

$$0 = f'(x_i)(x_r - x_{i+1}) + \frac{f''(\xi)}{2!}(x_r - x_i)^2 \qquad \text{(B6.2.4)}$$

Now, realize that the error is equal to the discrepancy between x_{i+1} and the true value x_r, as in

$$E_{t,i+1} = x_r - x_{i+1}$$

and Eq. (B6.2.4) can be expressed as

$$0 = f'(x_i)E_{t,i+1} + \frac{f''(\xi)}{2!}E_{t,i}^2 \qquad \text{(B6.2.5)}$$

If we assume convergence, both x_i and ξ should eventually be approximated by the root x_r, and Eq. (B6.2.5) can be rearranged to yield

$$E_{t,i+1} = \frac{-f''(x_r)}{2f'(x_r)}E_{t,i}^2 \qquad \text{(B6.2.6)}$$

According to Eq. (B6.2.6), the error is roughly proportional to the square of the previous error. This means that the number of correct decimal places approximately doubles with each iteration. Such behavior is referred to as *quadratic convergence*. Example 6.4 manifests this property.

GENERAL ANALYSIS

1)

Input data	x^4 + 2*x^3 – 5*x^2 -10 L=1 U=4 g=3				
Pression	.00001	.05	.005	.0005	.00005
newton	6	4	5	5	6
secant	14	5	12	13	14
Fixed-point	diverge	Diverge	Diverge	Diverge	Diverge
Brige-vita	6	4	5	5	6
bisection	17	5	8	11	15
False-posi-tion	11	5	6	8	9

2)

Input data	exp(x)-x L=0 U=1 G=1				
Precision	.00001	.05	.005	.0005	.00005
newton	4	3	3	3	4
secant	5	3	4	4	4
Fixed-point	22	7	11	15	19
Brige-vita	-	-	-	-	-

bisection	12	4	7	10	12
False-position	6	2	3	4	5

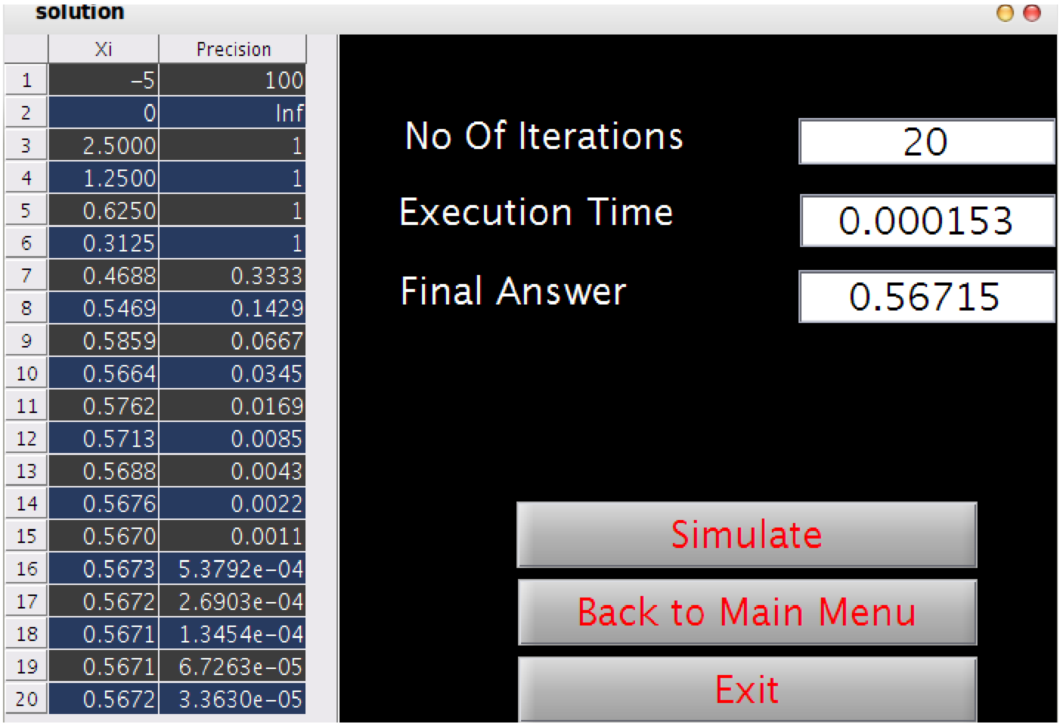
3)

Input data	2*x^2-8*x-24 L=5 U=10 G=7				
Precision	.00001	.05	.005	.0005	.00005
newton	4	2	3	3	4
secant	6	2	4	5	5
Fixed-point	Diverge	Diverge	Diverge	Diverge	Diverge
Brige-vita	4	2	3	3	4
bisection	17	5	8	11	15
False-position	10	2	4	11	9

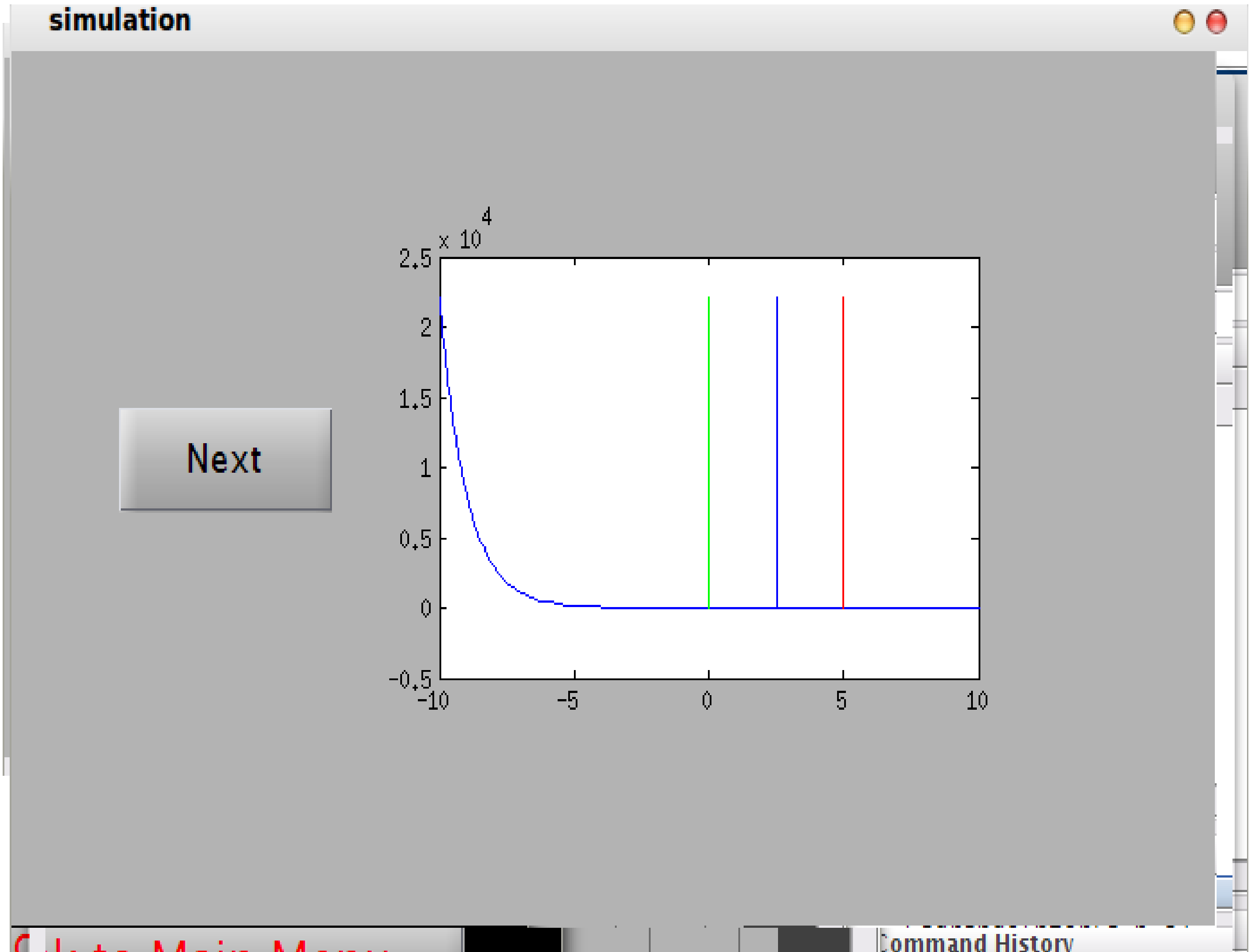
SAMPLE RUNS

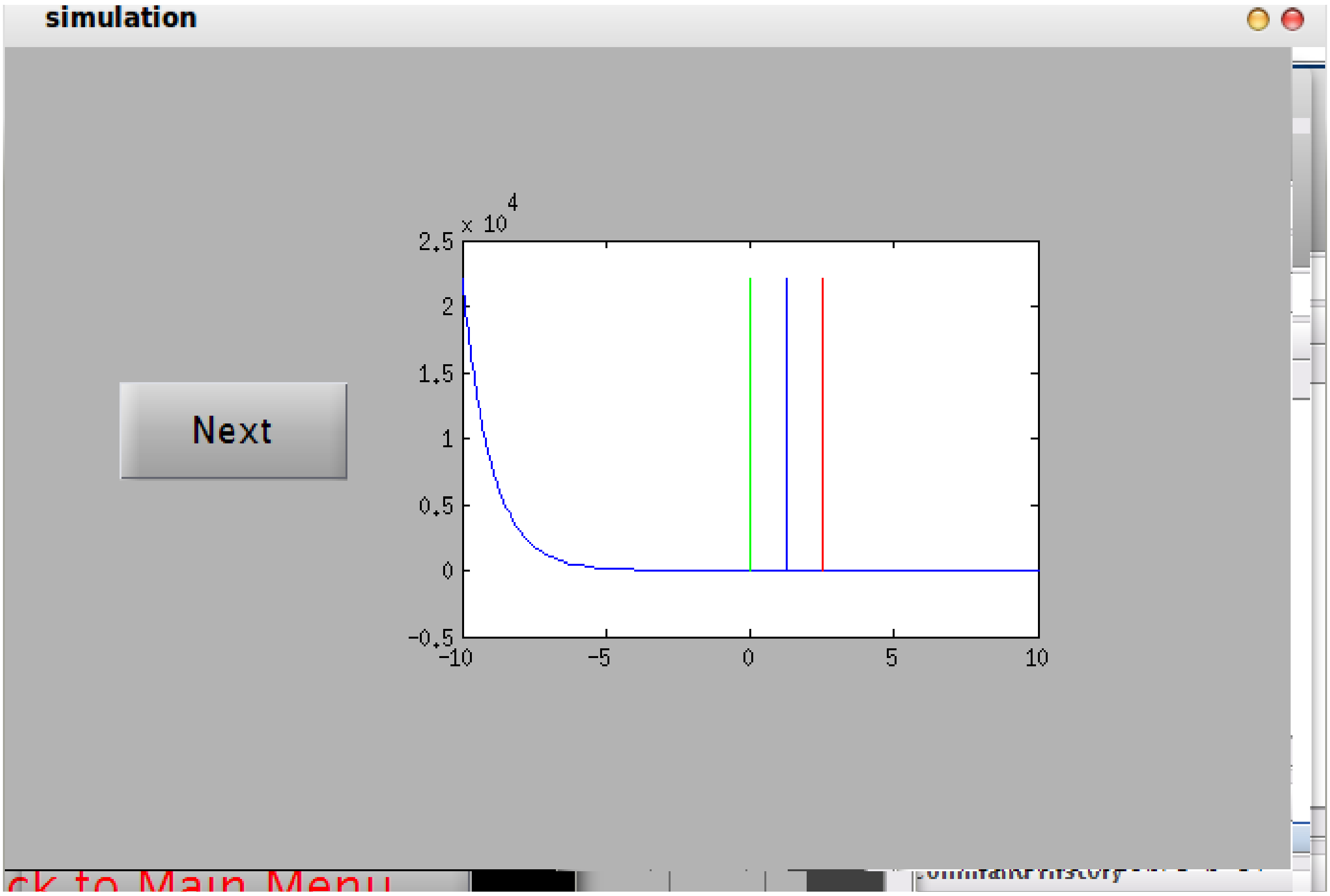
BISECTION

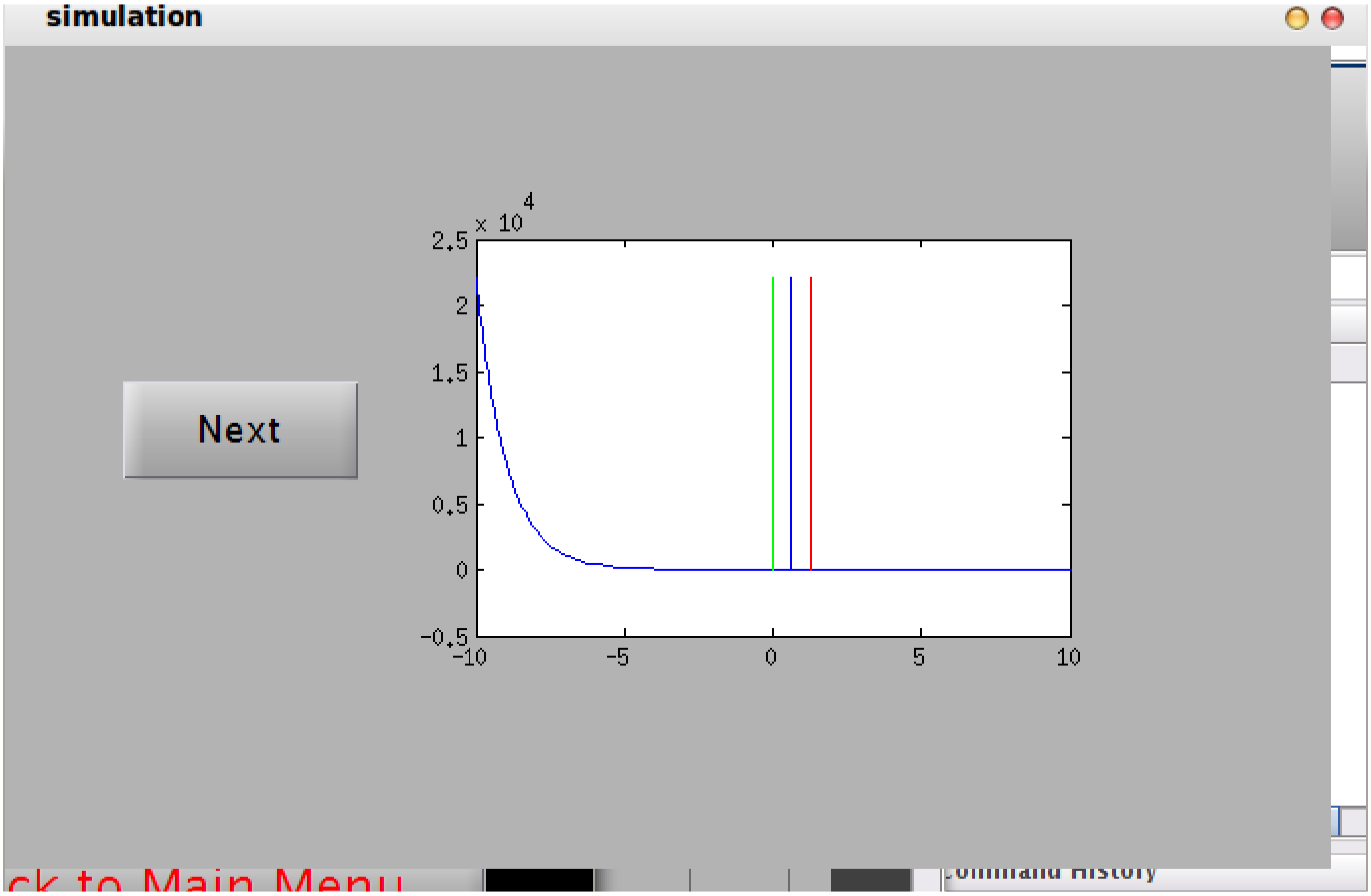
Input: exp(-x)-x , reading from file

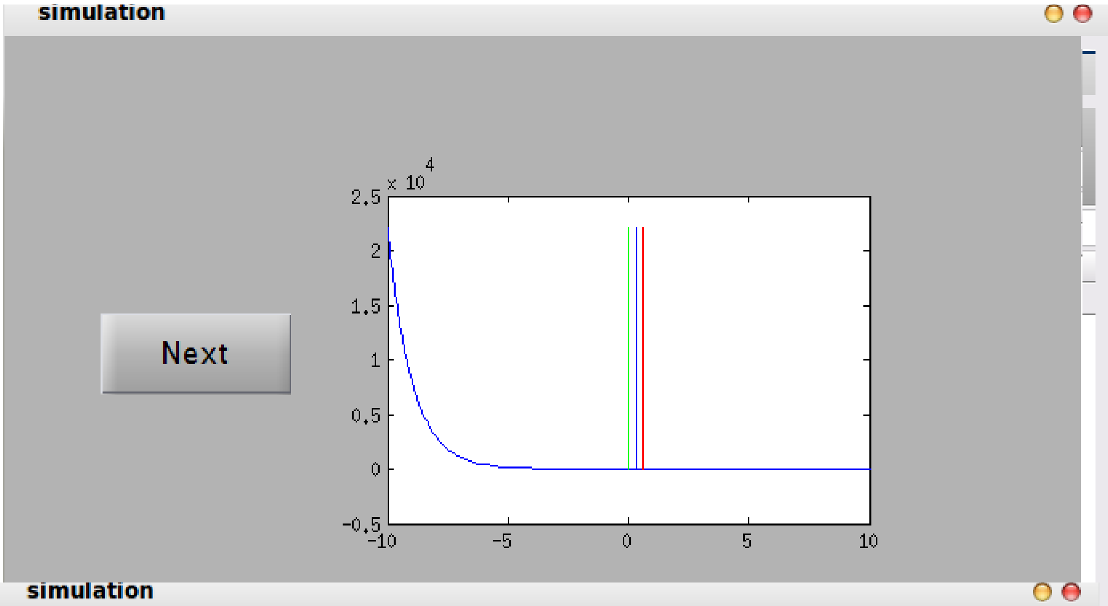


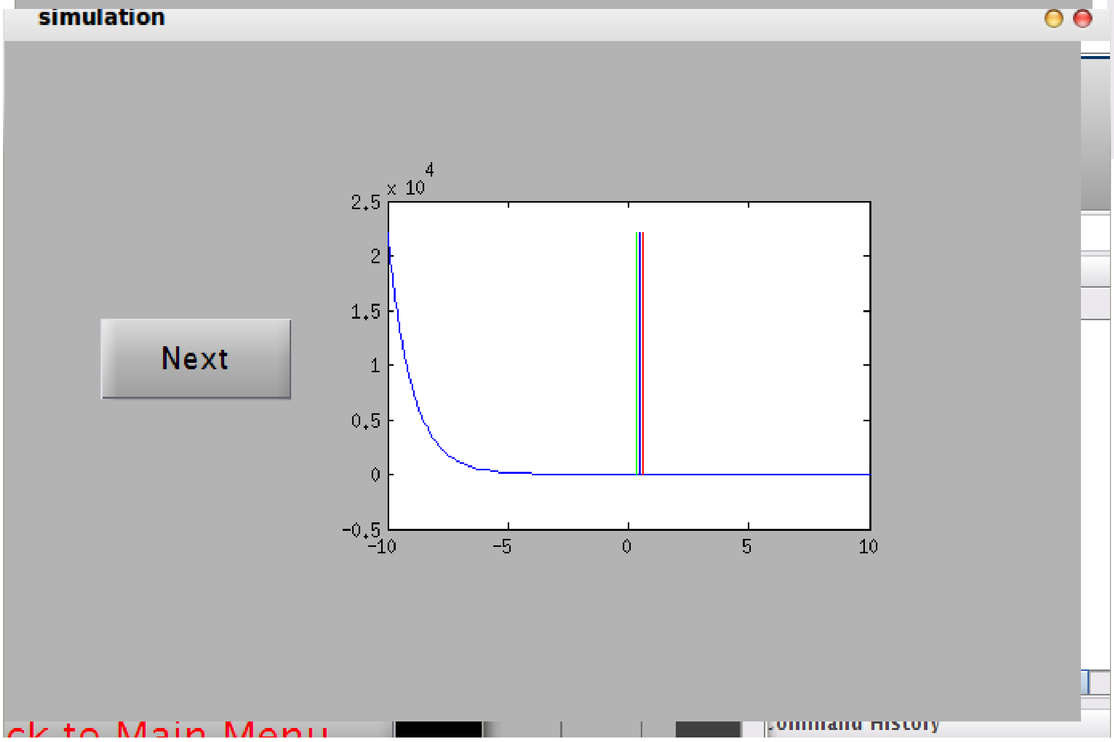
	Xi	Precision
1	-5	100
2	0	Inf
3	2.5000	1
4	1.2500	1
5	0.6250	1
6	0.3125	1
7	0.4688	0.3333
8	0.5469	0.1429
9	0.5859	0.0667
10	0.5664	0.0345
11	0.5762	0.0169
12	0.5713	0.0085
13	0.5688	0.0043
14	0.5676	0.0022
15	0.5670	0.0011
16	0.5673	5.3792e-04
17	0.5672	2.6903e-04
18	0.5671	1.3454e-04
19	0.5671	6.7263e-05
20	0.5672	3.3630e-05

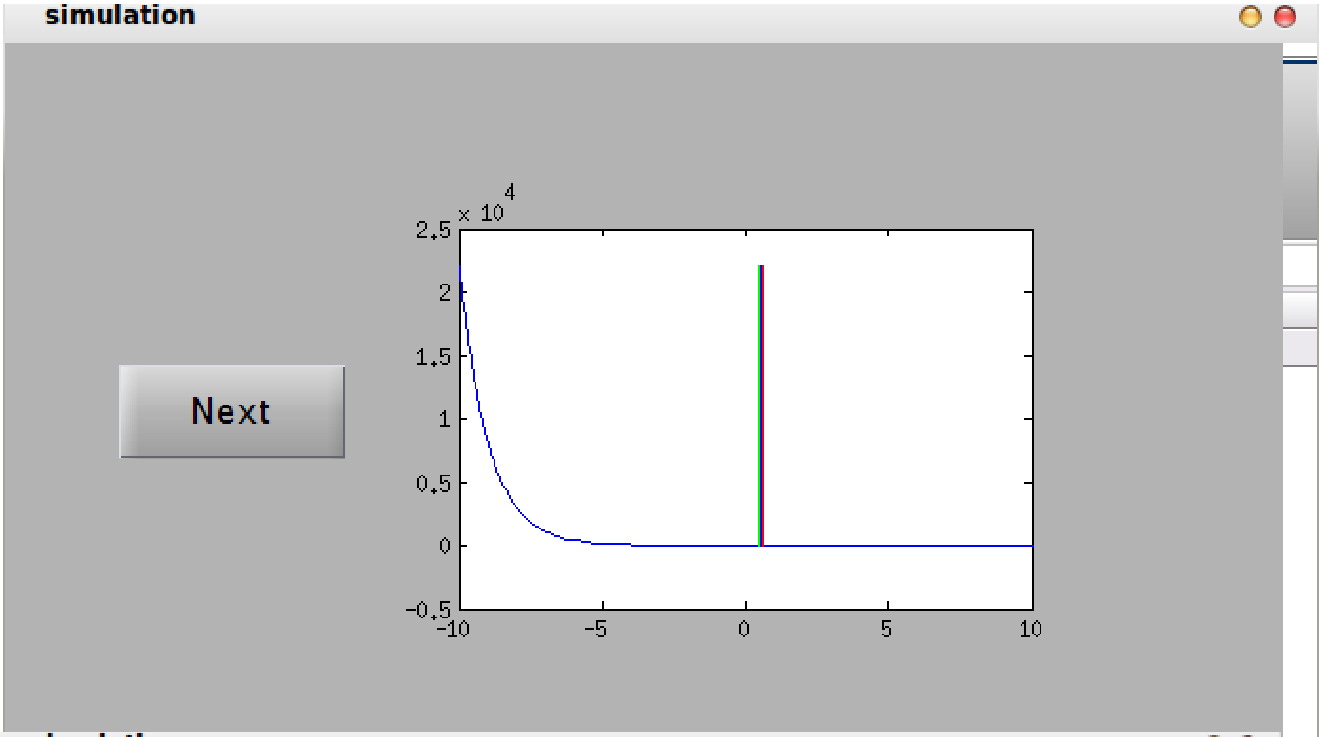


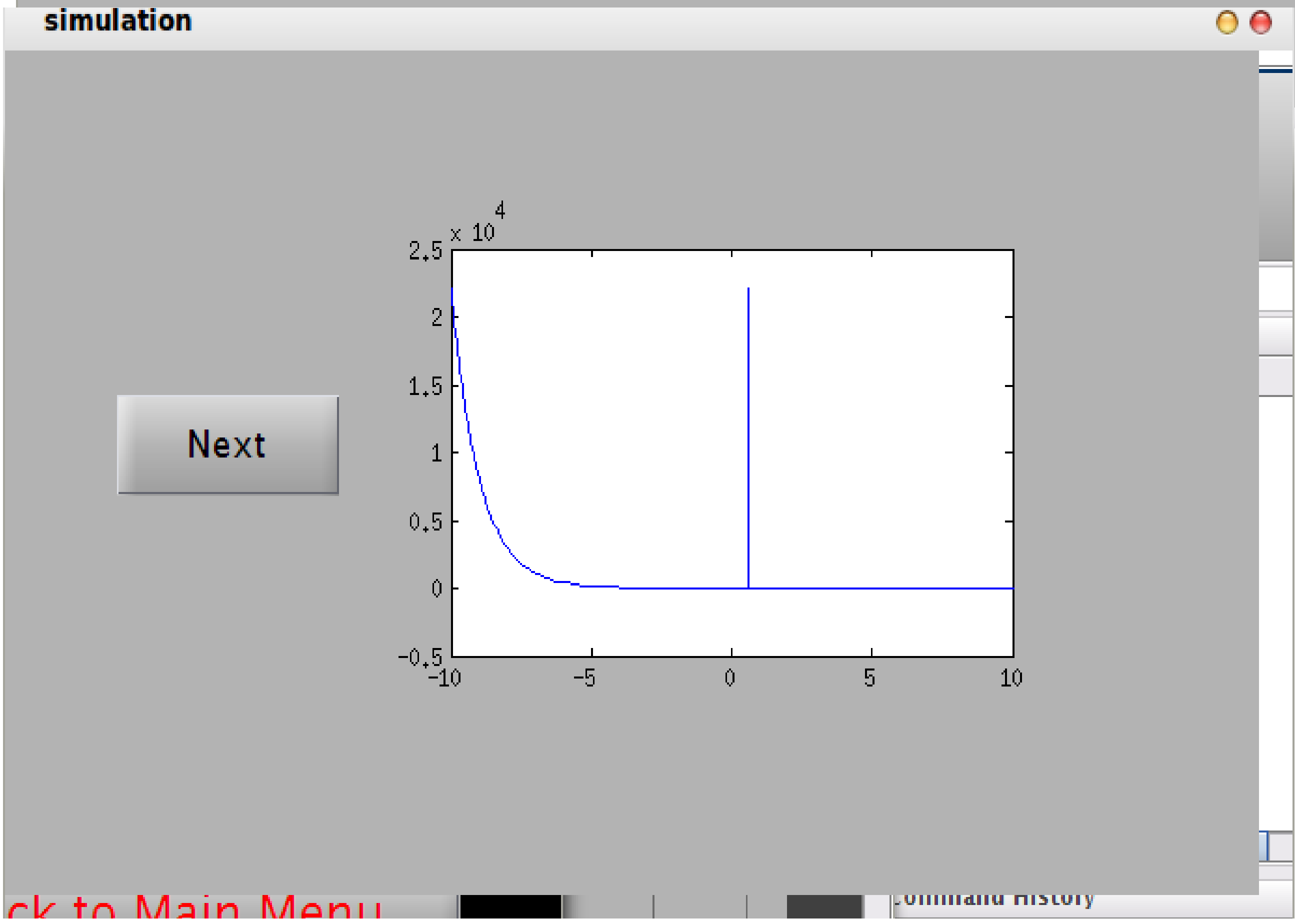

simulation
Next
x 10^4
2.5
2
1.5
1
0.5
0
-0.5
-10
-5
0
5
10
ck to Main Menu


simulation
Next
x 10^4
2.5
2
1.5
1
0.5
0
-0.5
-10
-5
0
5
10
ck to Main Menu


simulation
Next
x 10^4
2.5
2
1.5
1
0.5
0
-0.5
-10
-5
0
5
10


simulation
Next
x 10^4
2.5
2
1.5
1
0.5
0
-0.5
-10
-5
0
5
10

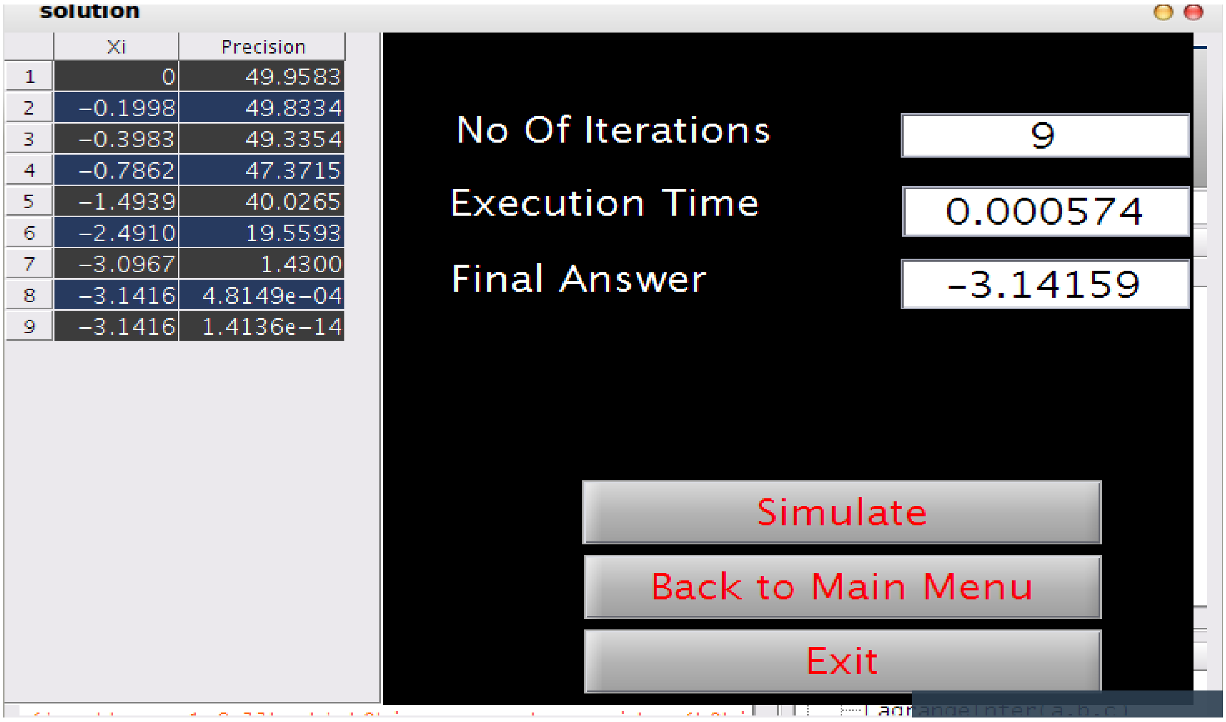

simulation
Next
x 10
4
2.5
2
1.5
1
0.5
0
-0.5
-10
-5
0
5
10

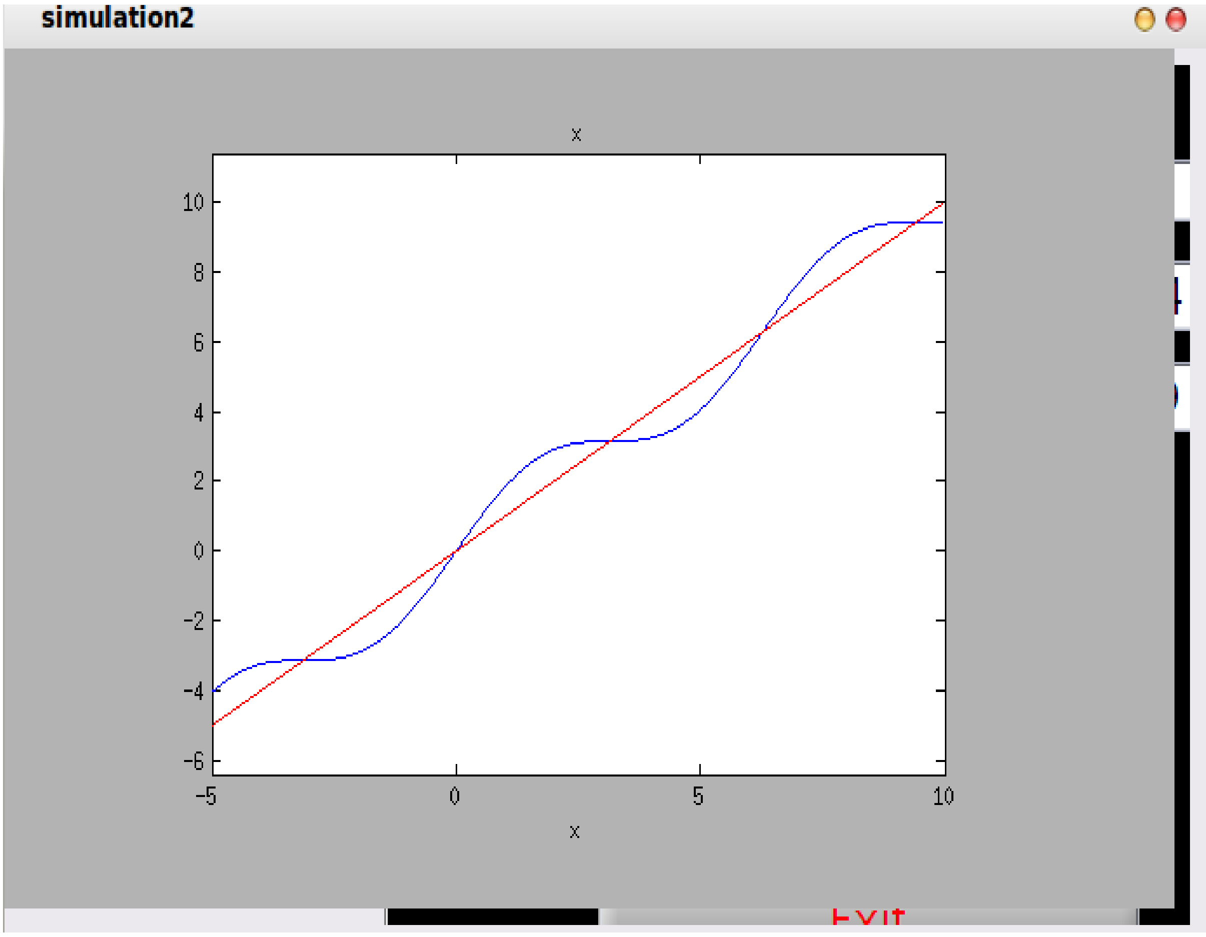

simulation
Next
x 10
4
2.5
2
1.5
1
0.5
0
-0.5
-10
-5
0
5
10

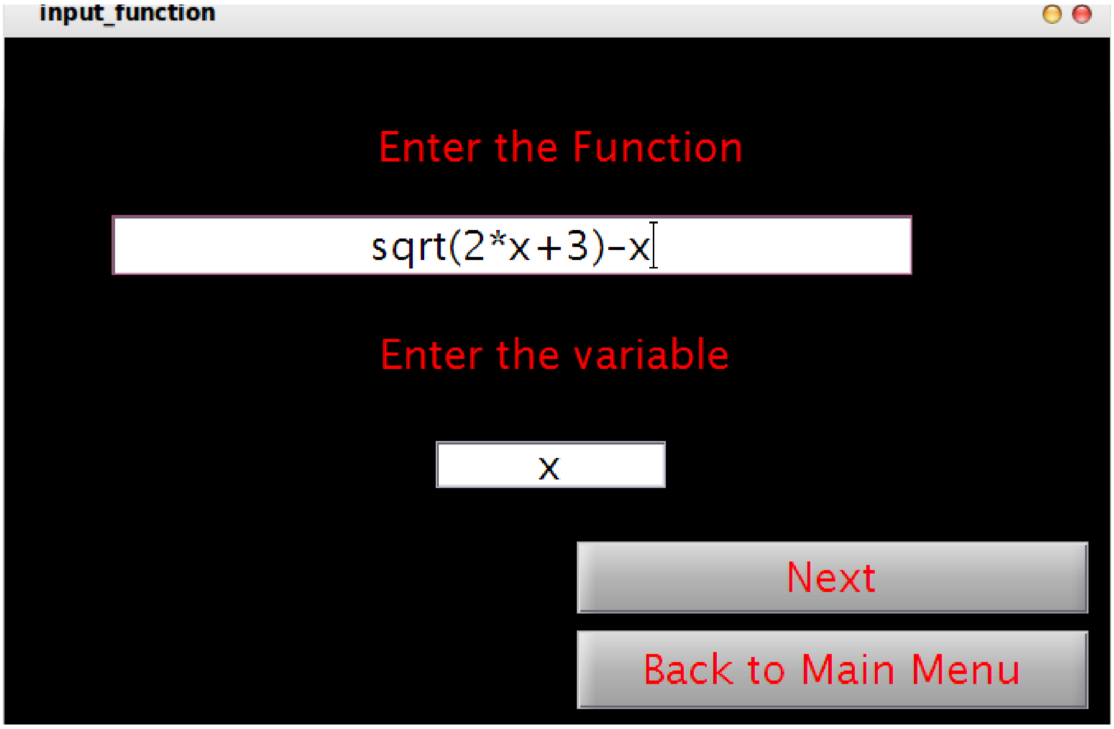
FIXED-POINT

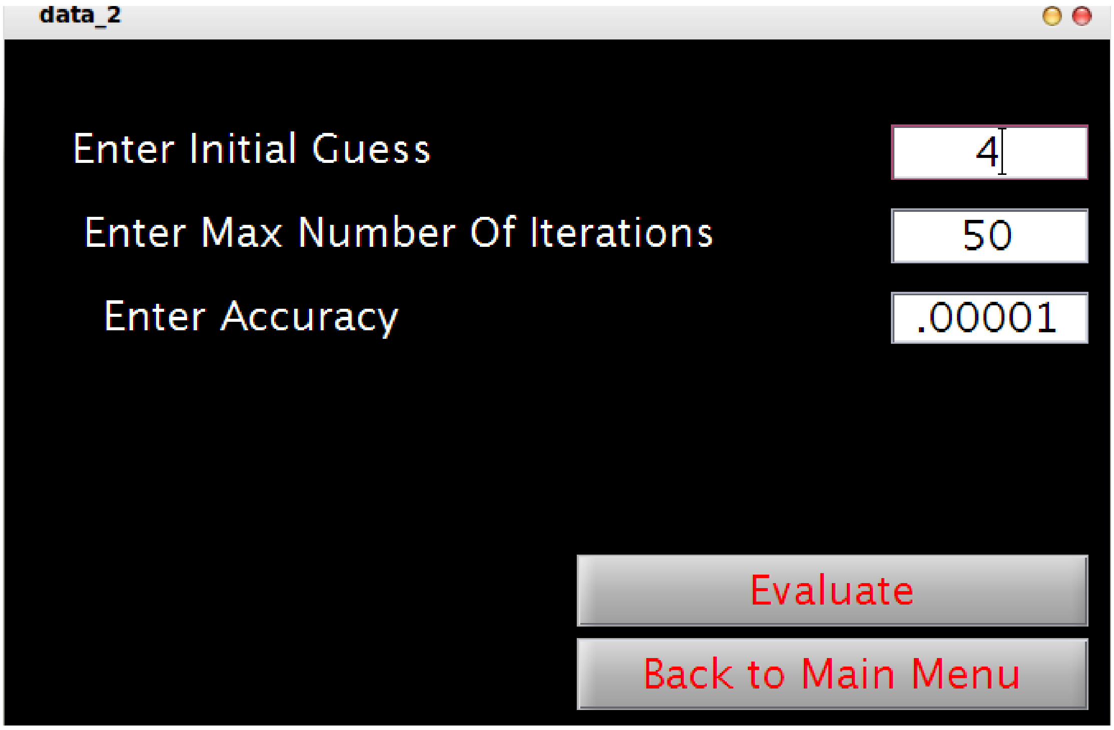
input function: sin(x)-x

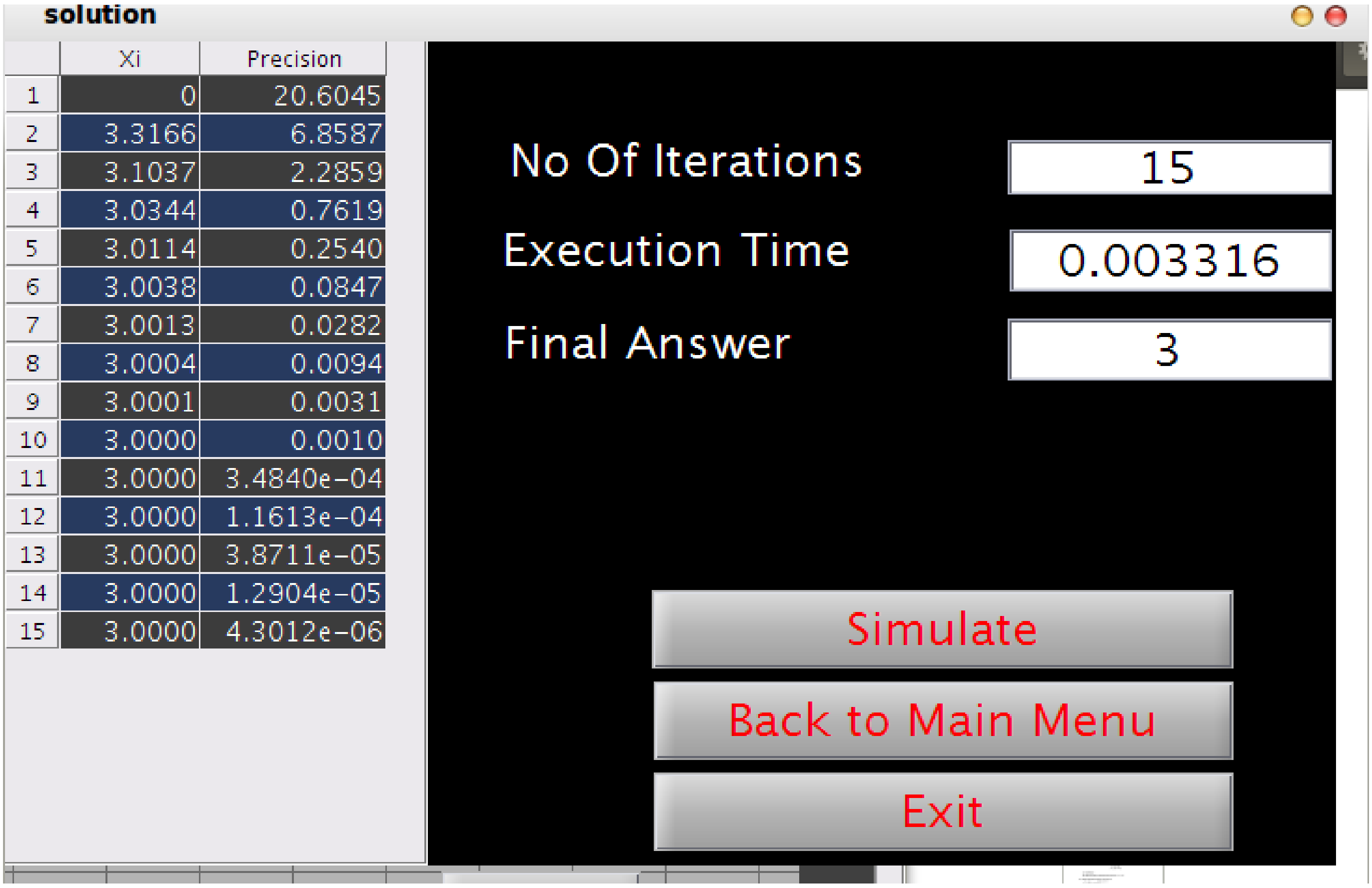
initial guess -1



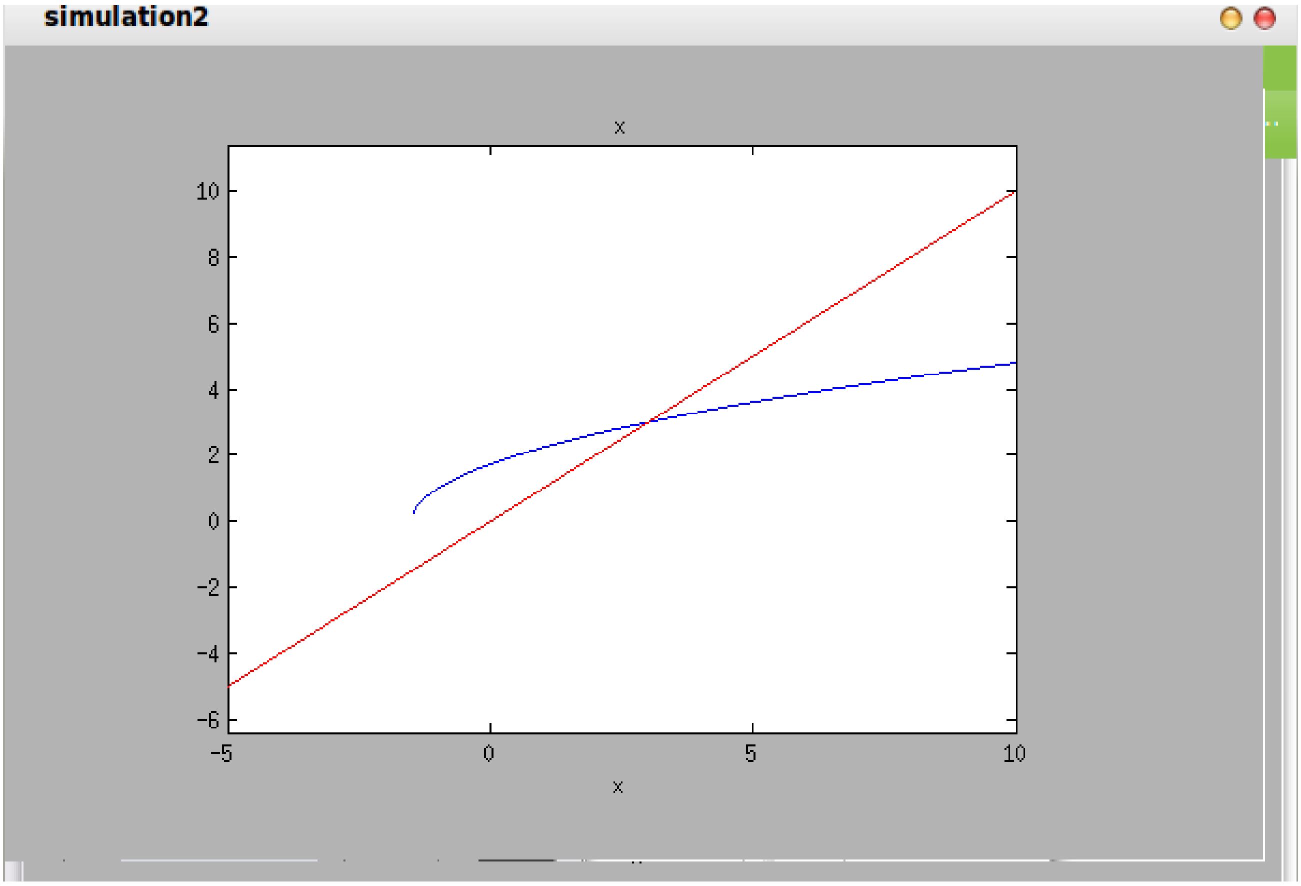


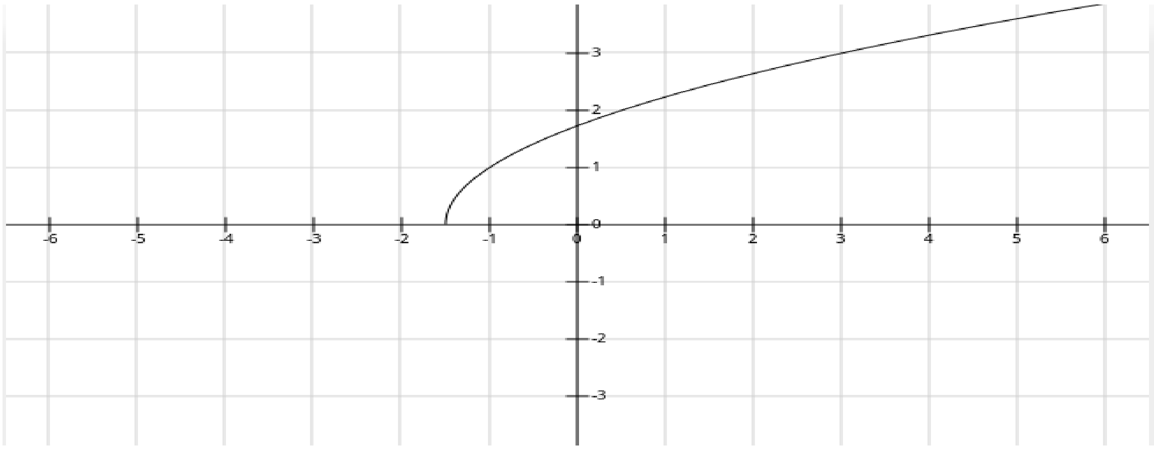

input_function
Enter the Function
sqrt(2*x+3)-x
Enter the variable
x
Next
Back to Main Menu


data_2
Enter Initial Guess
4
Enter Max Number Of Iterations
50
Enter Accuracy
.00001
Evaluate
Back to Main Menu

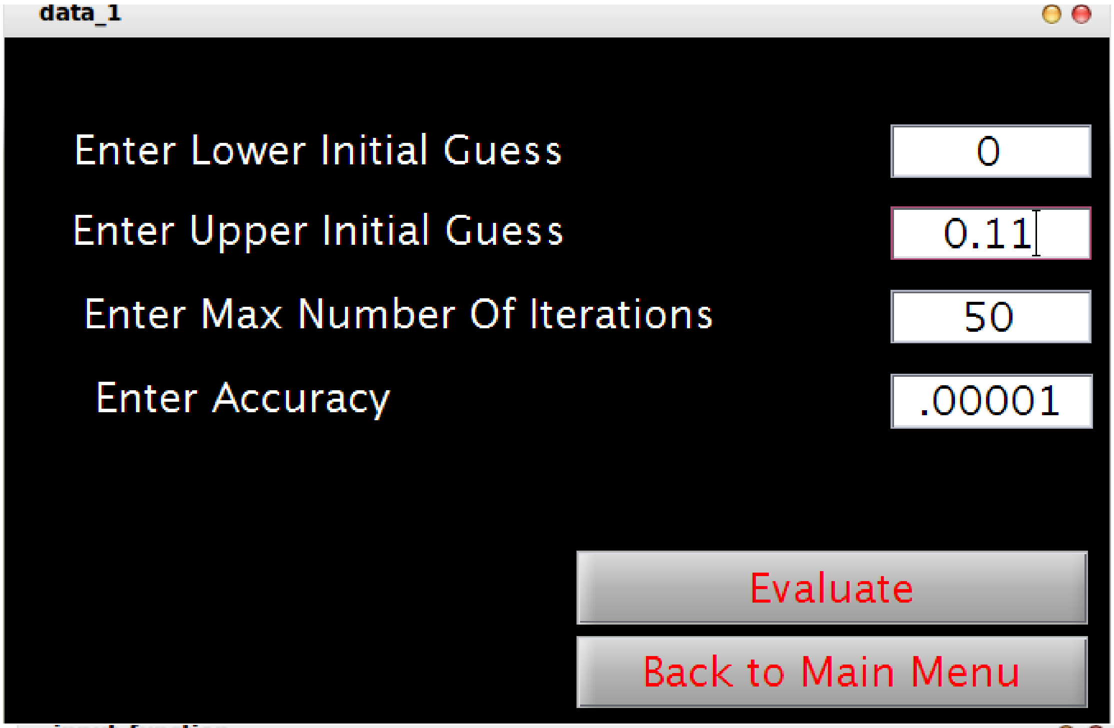


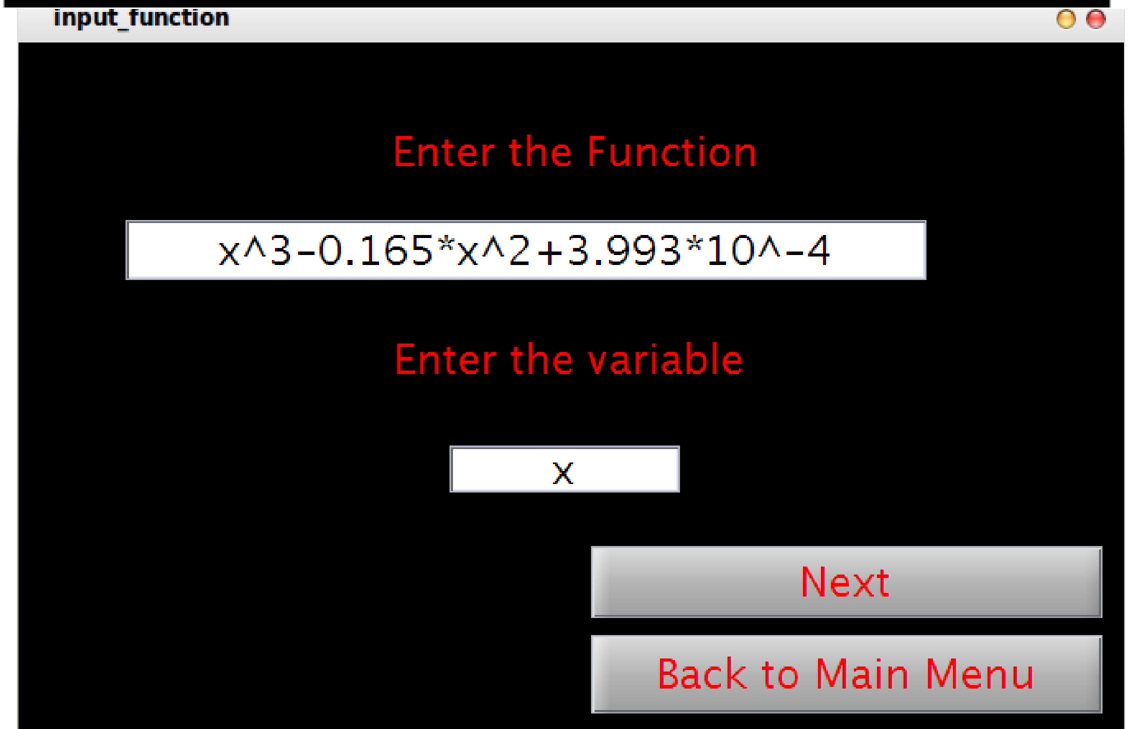
	Xi	Precision
1	0	20.6045
2	3.3166	6.8587
3	3.1037	2.2859
4	3.0344	0.7619
5	3.0114	0.2540
6	3.0038	0.0847
7	3.0013	0.0282
8	3.0004	0.0094
9	3.0001	0.0031
10	3.0000	0.0010
11	3.0000	3.4840e-04
12	3.0000	1.1613e-04
13	3.0000	3.8711e-05
14	3.0000	1.2904e-05
15	3.0000	4.3012e-06

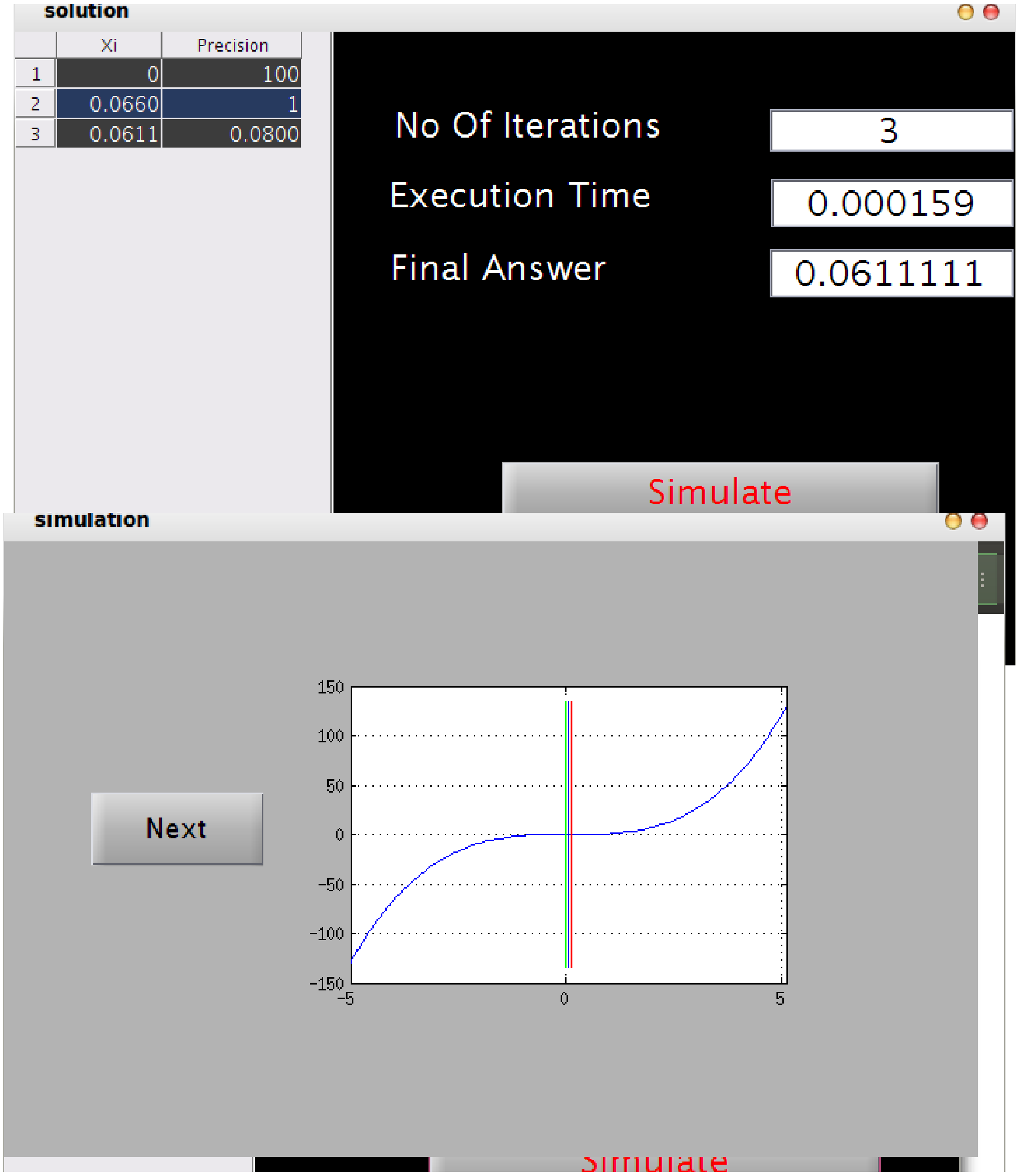


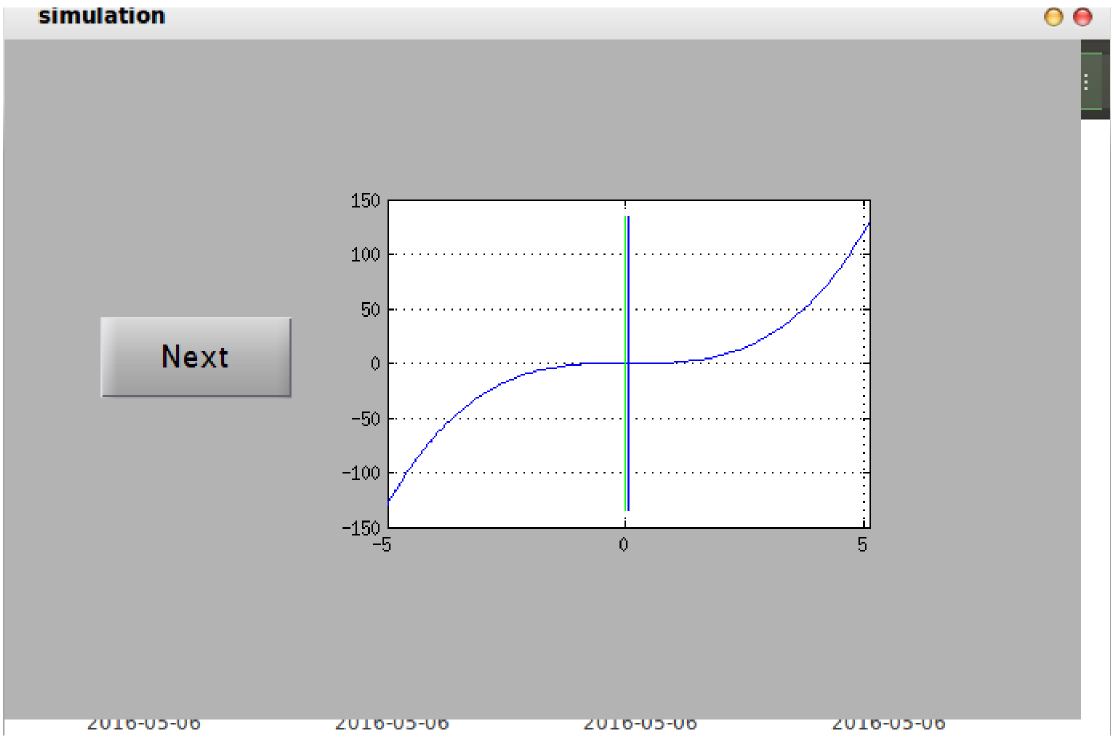


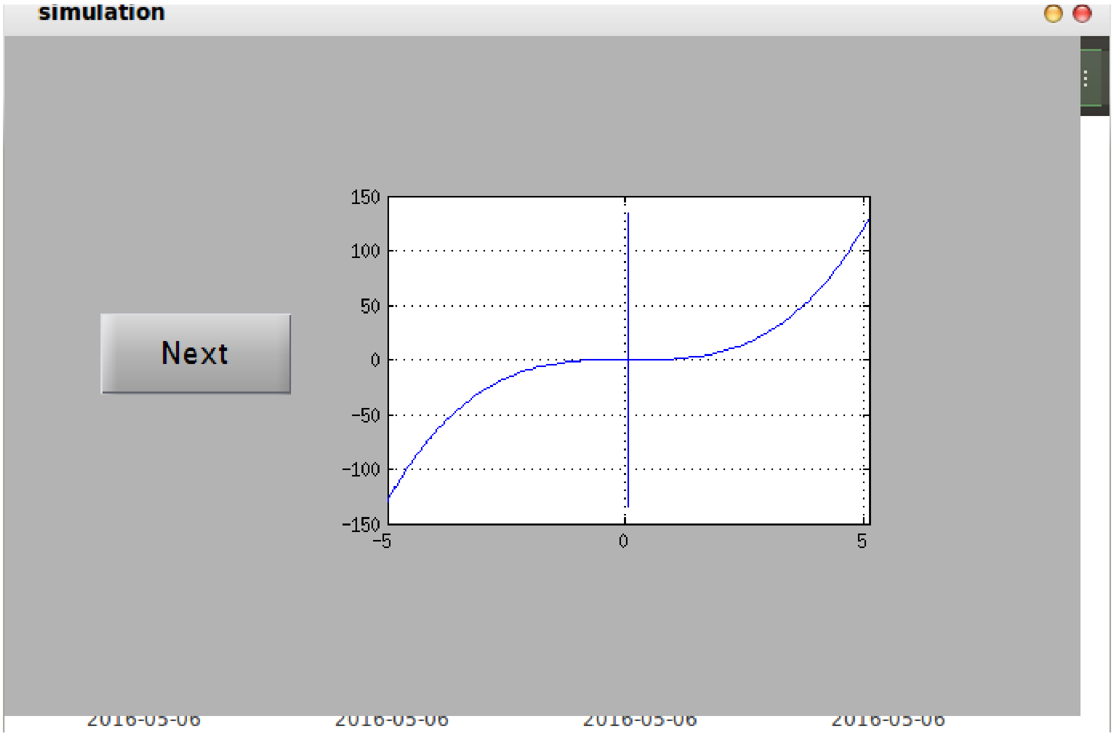
FALSE-POSITION





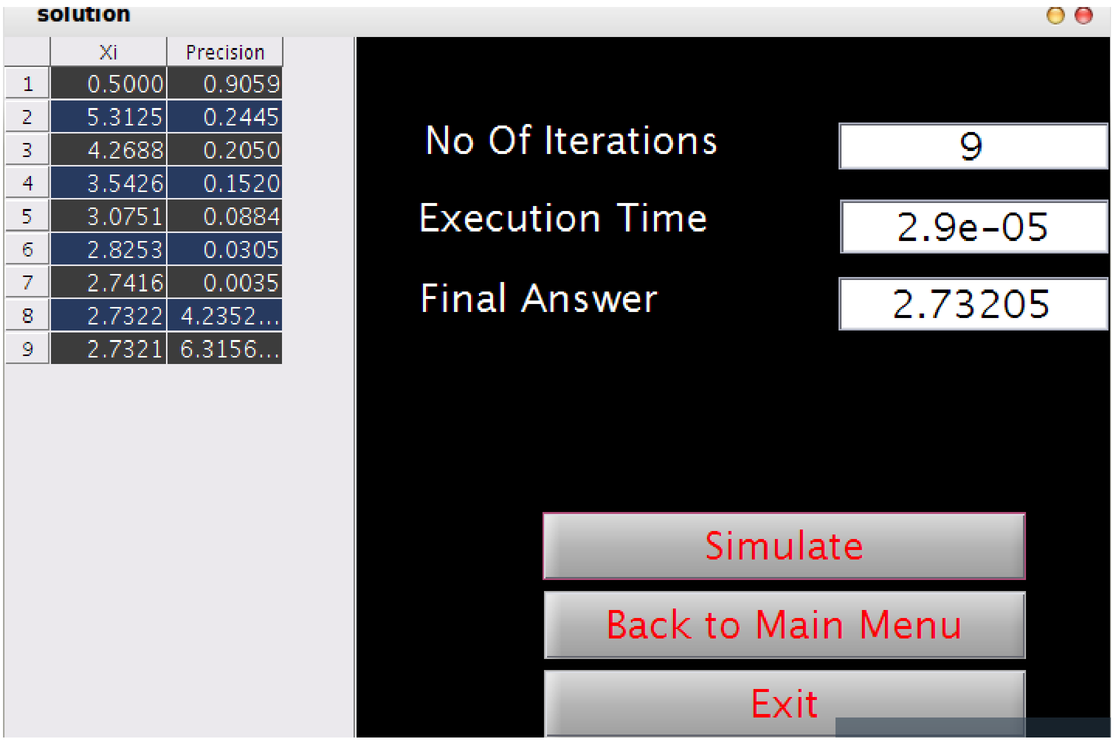

solution
Xi
Precision
1 0 100
2 0.0660 1
3 0.0611 0.0800
No Of Iterations
3
Execution Time
0.000159
Final Answer
0.0611111
Simulate
simulation
Next
150
100
50
0
-50
-100
-150
-5
0
5

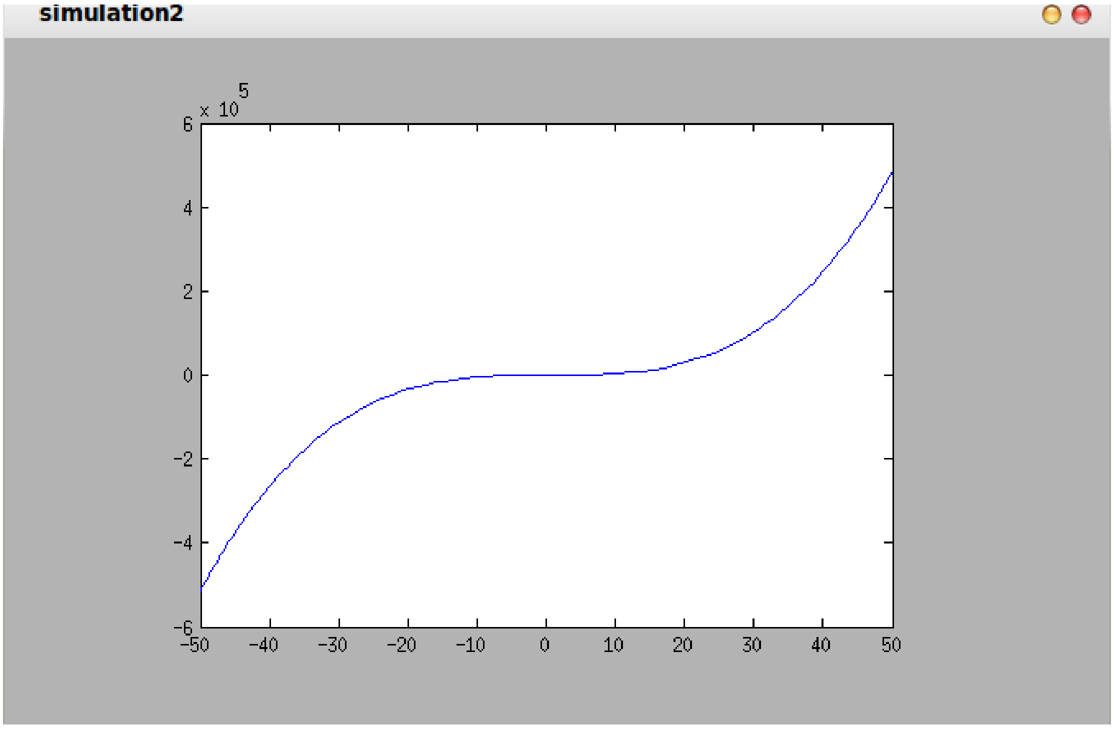

simulation
Next
150
100
50
0
-50
-100
-150
-5
0
5



BIRGE-VIETA

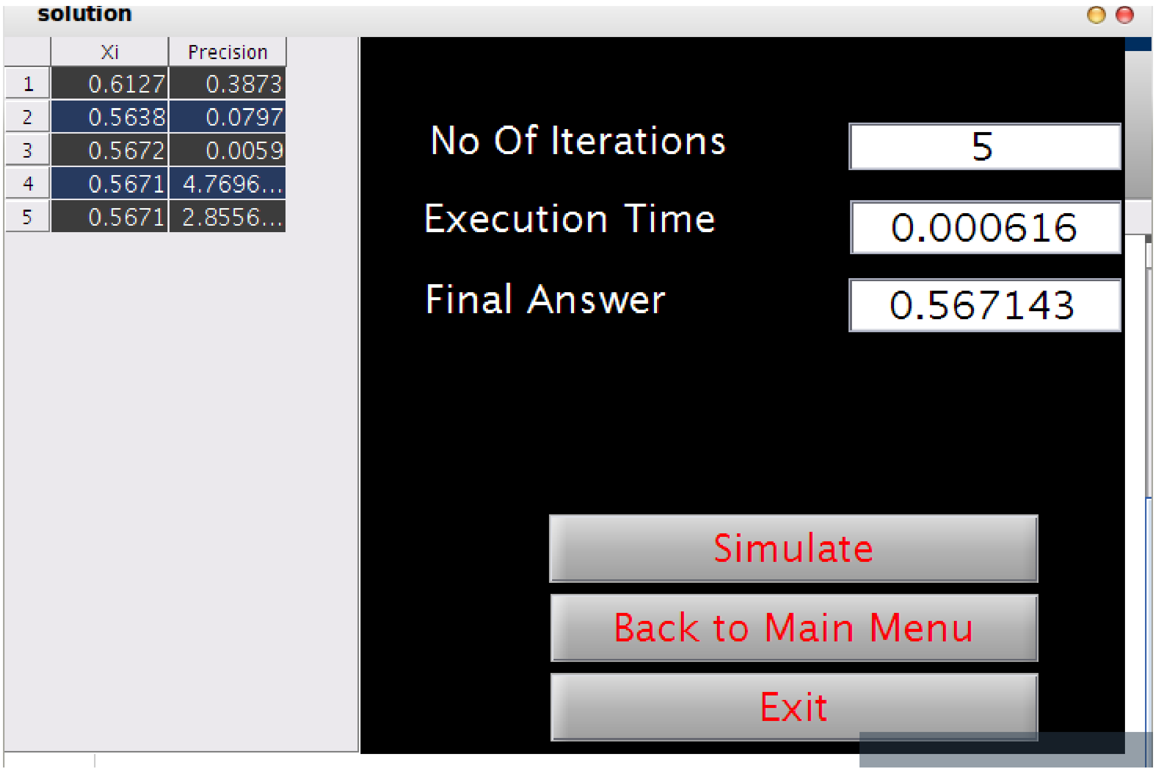
Input equation: x^4-2*x^3-4*x^2+4*x+4

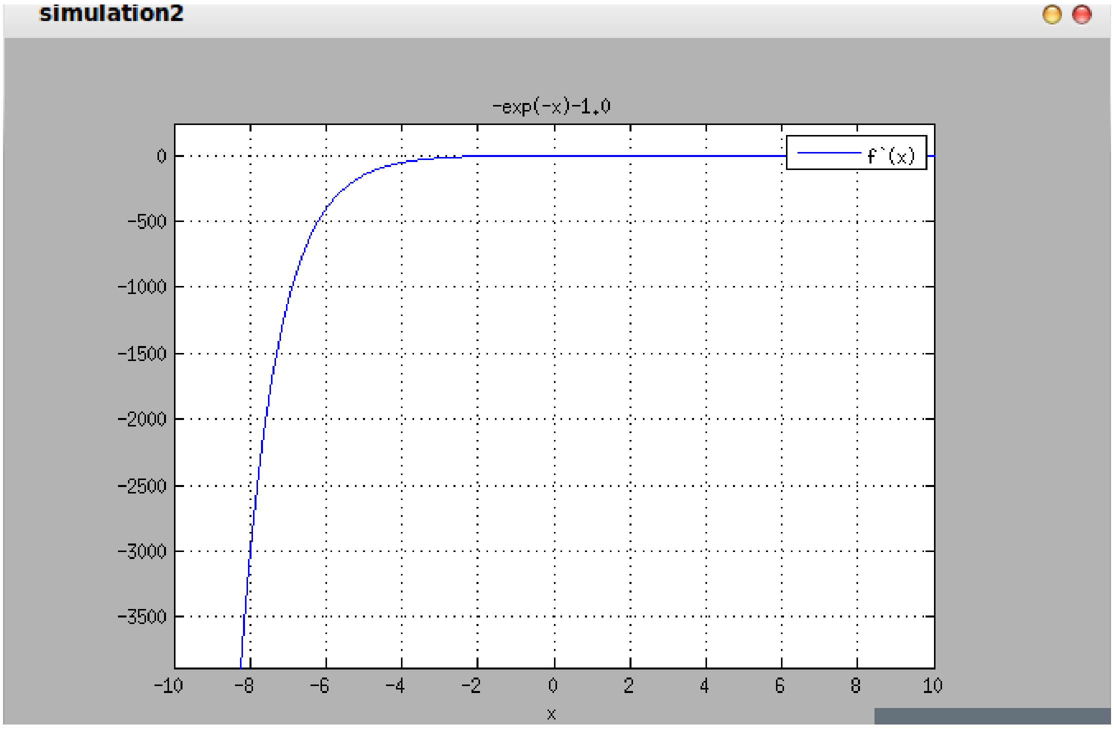




SECANT

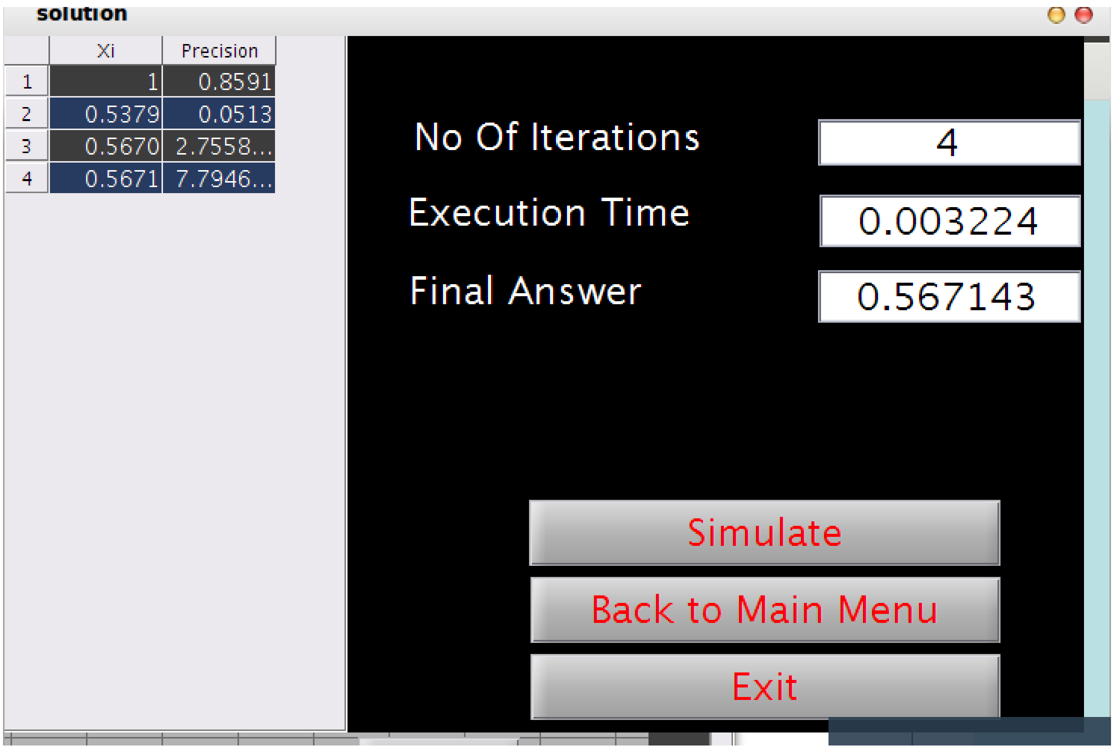
Input function: default





NEWTON

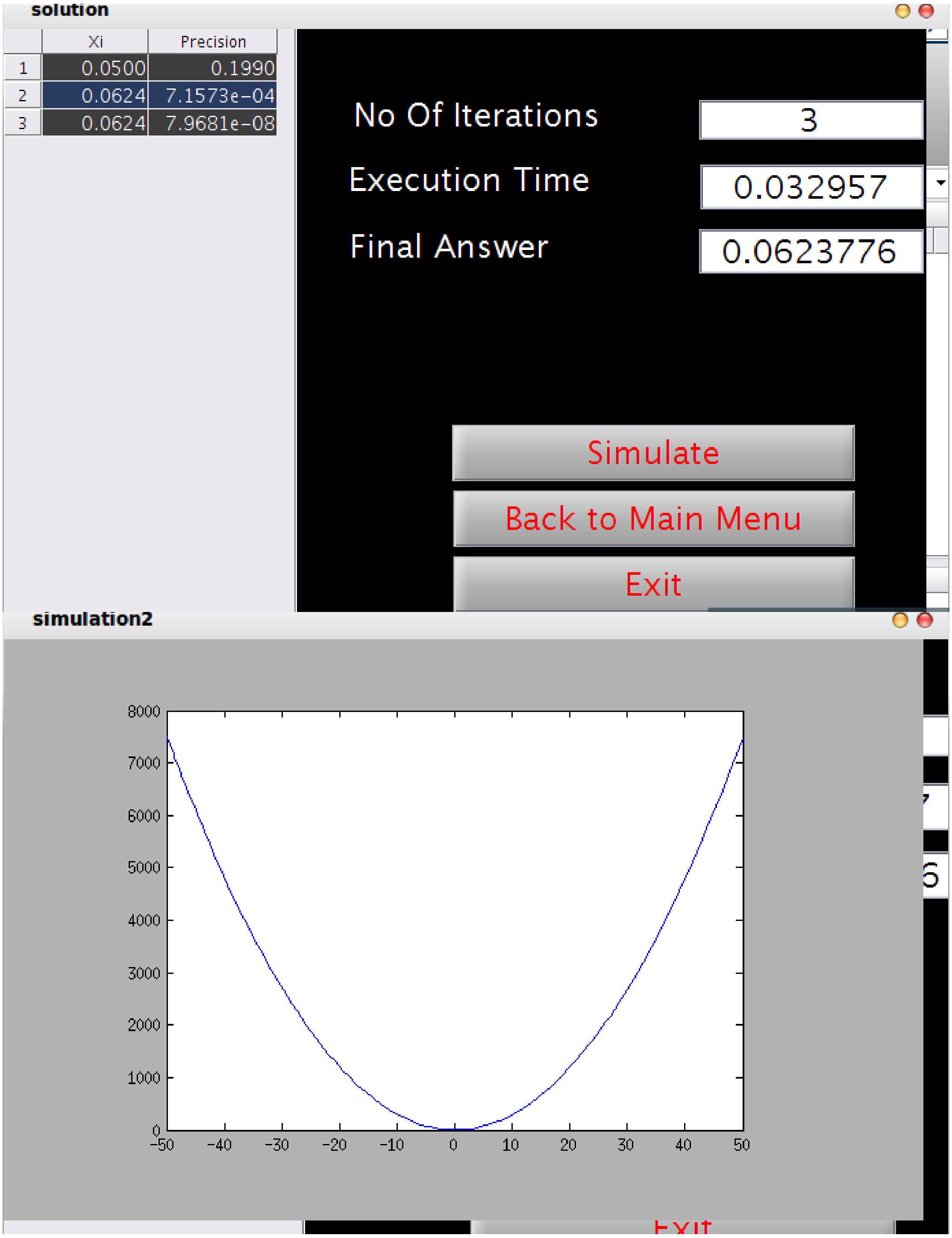
Input equation: default



READ FROM FILE

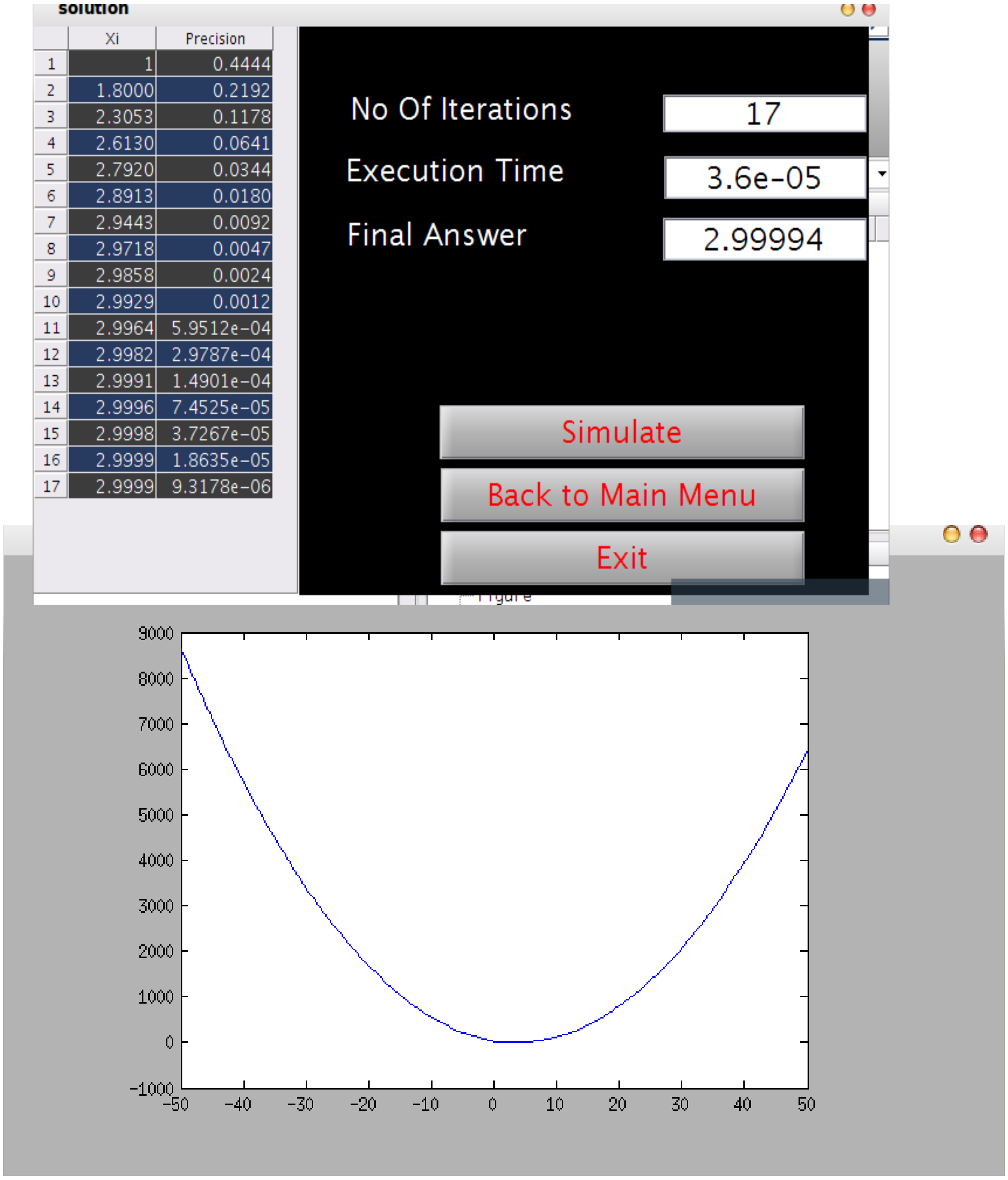
Input file (Newton raphson)

```
x^3-0.165*x^2+3.993*(10^-4)
4 0.05
```



input file: (Birge vieta)

```
x^3-11*x^2+39*x-45
6 1
```



PART TWO

PROBLEM STATEMENT

The aim of this assignment is to compare and analyze the behavior of numerical methods studied in class: Newton interpolation and Lagrange interpolation.

You are required to implement a program for querying the values of specific points using interpolation which takes as an input the polynomial order, sample point(s), corresponding value(s), the interpolation technique to use (Newton – Lagrange) and the query point(s).

PSEUDOCODE

NEWTON INTERPOLATION

```
SUBROUTINE NewtInt (x, y, n, xi, yint, ea)
  LOCAL fdd_{n,n}
  DOFOR i = 0, n
    fdd_{i,0} = y_i
  END DO
  DOFOR j = 1, n
    DOFOR i = 0, n − j
      fdd_{i,j} = (fdd_{i+1,j−1} − fdd_{i,j−1})/(x_{i+j} − x_i)
    END DO
  END DO
  xterm = 1
  yint_0 = fdd_{0,0}
  DOFOR order = 1, n
    xterm = xterm * (xi − x_{order−1})
    yint2 = yint_{order−1} + fdd_{0,order} * xterm
    ea_{order−1} = yint2 − yint_{order−1}
    yint_{order} = yint2
  END order
END NewtInt
```

LAGRANGE INTERPOLATION

```
function  Lagrange
   returns  yi, time, fun, Min, Max
   paramters x, y, xi
    ni = xi.length
    n =  x.length
    Min = x[1] - 0.1
    Max = x[end] + 0.1
    start timer
    for k from 1 to ni
        for i from  1 to n
              l <- 1
```

```
            for j from 1 to n
                if j not equal i
                 l <- l .* ((X - x(j)) / (x(i) - x(j)))
                end
            end
            fun[k] <- fun[k] + l * y[i]
        end
        yi(k) <-  evaluate fun[k] for xi[k]
    end
    time = end timer;
end
```

DATA STRUCTURE

Vector of x and y for points input in lagrange and newton interpolation and a vector for x input in the program required to find its f(x), a vector of f(x) of given xs returned after solving. Use array of symbols to solve functions.

ANALYSIS AND PROBLEMATIC FUNCTIONS

Given (n+1) points {(x0, y0), (x1, y1), …, (xn, yn)}, the points defined by $(x_i)_{0 \le i \le n}$ are called points of interpolation.
The points defined by (y_i) $0 \le i \le n$ are the values of interpolation. To interpolate a function f, the values of interpolation are defined as follows:

$$y_i = f(x_i), \text{ for all } i = 0, \ldots, n$$

NEWTON INTERPOLATION

Newton interpolation, you get the coefficients reasonably fast (quadratic time), the evaluation is much more stable (roughly because there is usually a single dominant term for a given xx), the evaluation can be done quickly and straightforwardly using Horner's method, and adding an additional node just amounts to adding a single additional term. It is also fairly easy to see how to interpolate derivatives using the Newton framework. Calculating the Divided Difference table is a complex operation.

LAGRANGE INTERPOLATION

Lagrange interpolation is mostly just useful for theory. Actually computing with it requires huge numbers and catastrophic cancellations. In floating point arithmetic this is very bad. It does have some small advantages: for instance, the Lagrange approach amounts to diagonalizing the problem of finding the coefficients, so it takes only linear time to find the coefficients. This is good if you need to use the same set of points repeatedly. But all of these advantages do not make up for the problems associated with trying to actually evaluate a Lagrange interpolating polynomial.

- The amount of computation required is large
- Interpolation for additional values of requires the same amount of effort as the first value (i.e. no part of the previous calculation can be used)
- When the number of interpolation points are changed (increased/decreased), the results of the previous computations cannot be used

E.1 Lagrange polynomials

We wish to find the polynomial interpolating the points

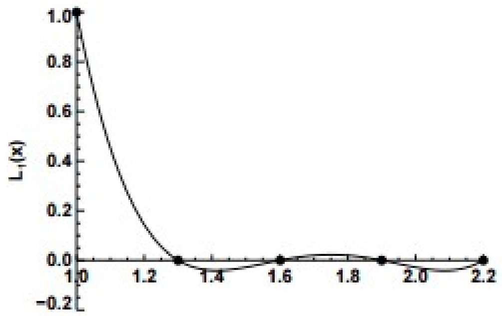
x	1	1.3	1.6	1.9	2.2
$f(x)$	0.1411	−0.6878	−0.9962	−0.5507	0.3115

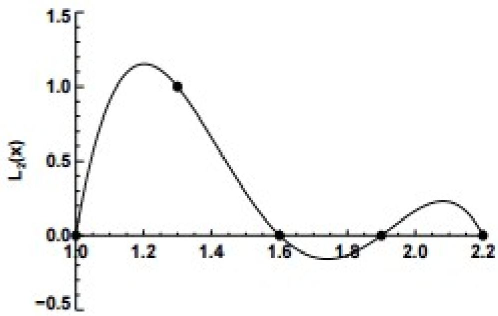
where $f(x) = \sin(3x)$, and estimate $f(1.5)$.

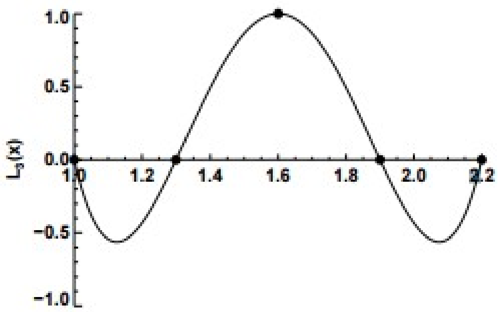
First, we find Lagrange polynomials $L_k(x)$, $k = 1 \ldots 5$,

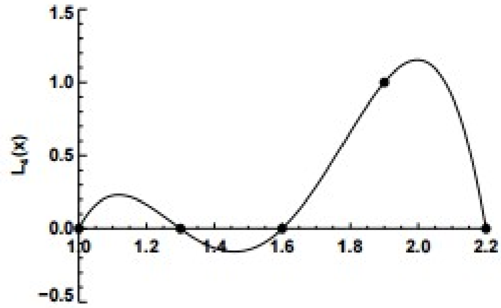
$$L_1(x) = \frac{(x-1.3)(x-1.6)(x-1.9)(x-2.2)}{(1-1.3)(1-1.6)(1-1.9)(1-2.2)}, \quad L_2(x) = \frac{(x-1)(x-1.6)(x-1.9)(x-2.2)}{(1.3-1)(1.3-1.6)(1.3-1.9)(1.3-2.2)}$$
$$L_3(x) = \frac{(x-1)(x-1.3)(x-1.9)(x-2.2)}{(1.6-1)(1.6-1.3)(1.6-1.9)(1.6-2.2)}, \quad L_4(x) = \frac{(x-1)(x-1.3)(x-1.6)(x-2.2)}{(1.9-1)(1.9-1.3)(1.9-1.6)(1.9-2.2)}$$
$$L_5(x) = \frac{(x-1)(x-1.3)(x-1.6)(x-1.9)}{(2.2-1)(2.2-1.3)(2.2-1.6)(2.2-1.9))}$$

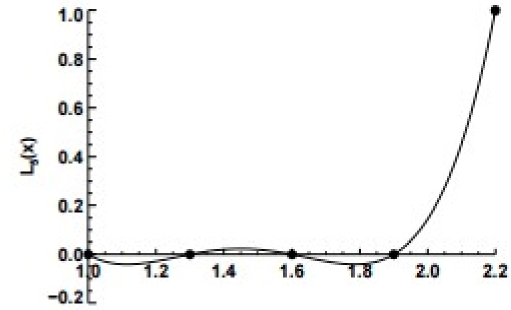
with the following graphs,





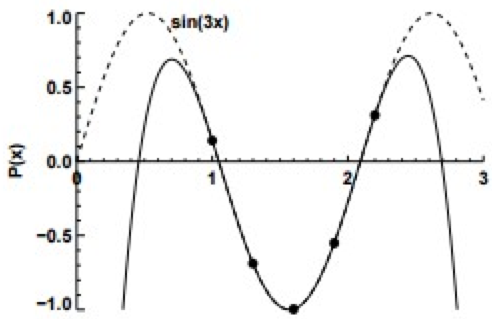






Clearly, $L_k(x_i) = \delta_{ik}$. Next, the polynomial approximation can be assembled,

$$P(x) = 0.1411 \times L_1(x) - 0.6878 \times L_2(x) - 0.9962 \times L_3(x) - 0.5507 \times L_4(x) + 0.3115 \times L_5(x).$$

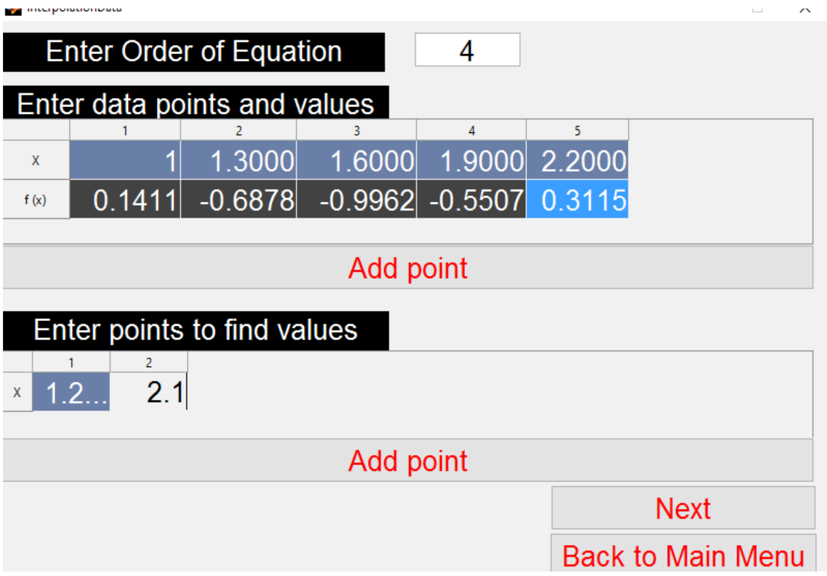


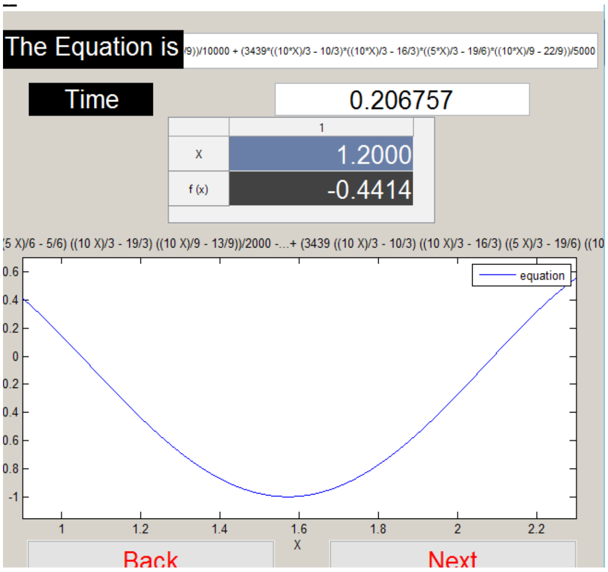
$x = 1.2$

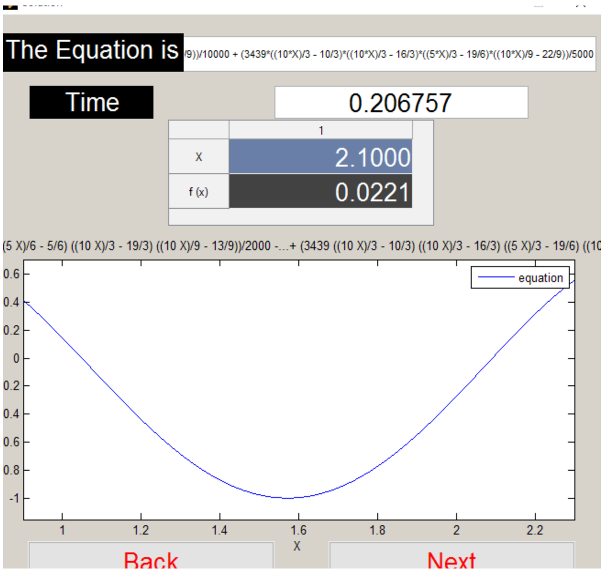
$P(1.2) = 0.1411 * L_1(1.2) - 80.6878 * L_2(1.2) - 0.9962 * L_3(1.2) - 0.9962 * L_3(1.2) - 0.5507 * L_4(1.2) + 0.3115 * L_5(1.2) = -0.4414$

$x = 2.1$

$P(2.1) = 0.1411 * L_1(2.1) - 80.6878 * L_2(2.1) - 0.9962 * L_3(2.1) - 0.9962 * L_3(2.1) - 0.5507 * L_4(2.1) + 0.3115 * L_5(2.1) = 0.0221$


Enter Order of Equation
4
Enter data points and values
1 2 3 4 5
x 1 1.3000 1.6000 1.9000 2.2000
f (x) 0.1411 -0.6878 -0.9962 -0.5507 0.3115
Add point
Enter points to find values
1 2
x 1.2... 2.1
Add point
Next
Back to Main Menu


The Equation is
9))/10000 + (3439*((10*X)/3 - 10/3)*((10*X)/3 - 16/3)*((5*X)/3 - 19/6)*((10*X)/9 - 22/9))/5000
Time
0.206757
1
x 1.2000
f (x) -0.4414
(5 X)/6 - 5/6) ((10 X)/3 - 19/3) ((10 X)/9 - 13/9))/2000 -...+ (3439 ((10 X)/3 - 10/3) ((10 X)/3 - 16/3) ((5 X)/3 - 19/6) ((10
equation
X
Back
Next


The Equation is
9))/10000 + (3439*((10*X)/3 - 10/3)*((10*X)/3 - 16/3)*((5*X)/3 - 19/6)*((10*X)/9 - 22/9))/5000
Time
0.206757
1
x 2.1000
f (x) 0.0221
(5 X)/6 - 5/6) ((10 X)/3 - 19/3) ((10 X)/9 - 13/9))/2000 -...+ (3439 ((10 X)/3 - 10/3) ((10 X)/3 - 16/3) ((5 X)/3 - 19/6) ((10
equation
X
Back
Next

SAMPLE RUNS

NEWTON INTERPOLATION

A robot arm with a rapid laser scanner is doing a quick quality check on holes drilled in a rectangular plate. The hole centers in the plate that describe the path the arm needs to take are given below.

If the laser is traversing from x = 2 to x = 4.25 in a linear path, find the value of y at x = 4 using the Newton's Divided Difference method for quadratic interpolation.

x (m)	y (m)
2	7.2
4.25	7.1
5.25	6.0
7.81	5.0
9.2	3.5
10.6	5.0

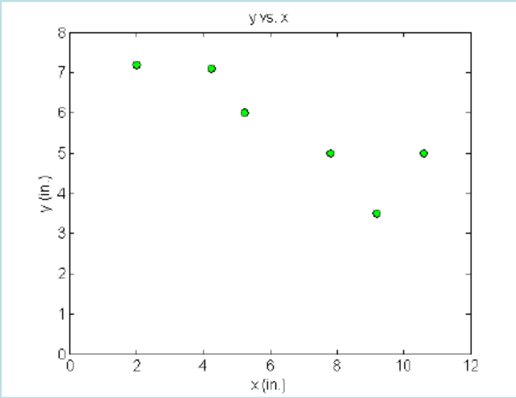


Figure 2 Location of holes on the

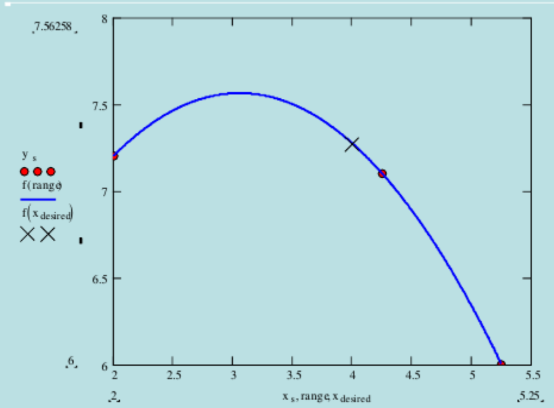
Quadratic Interpolation (cont.)

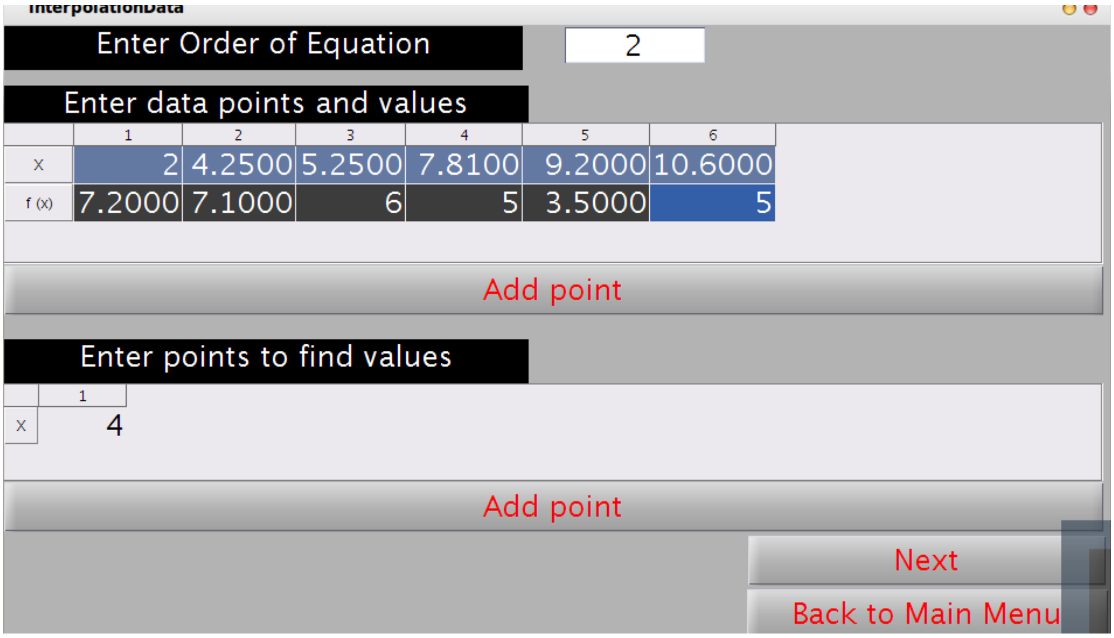
$$y(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1)$$

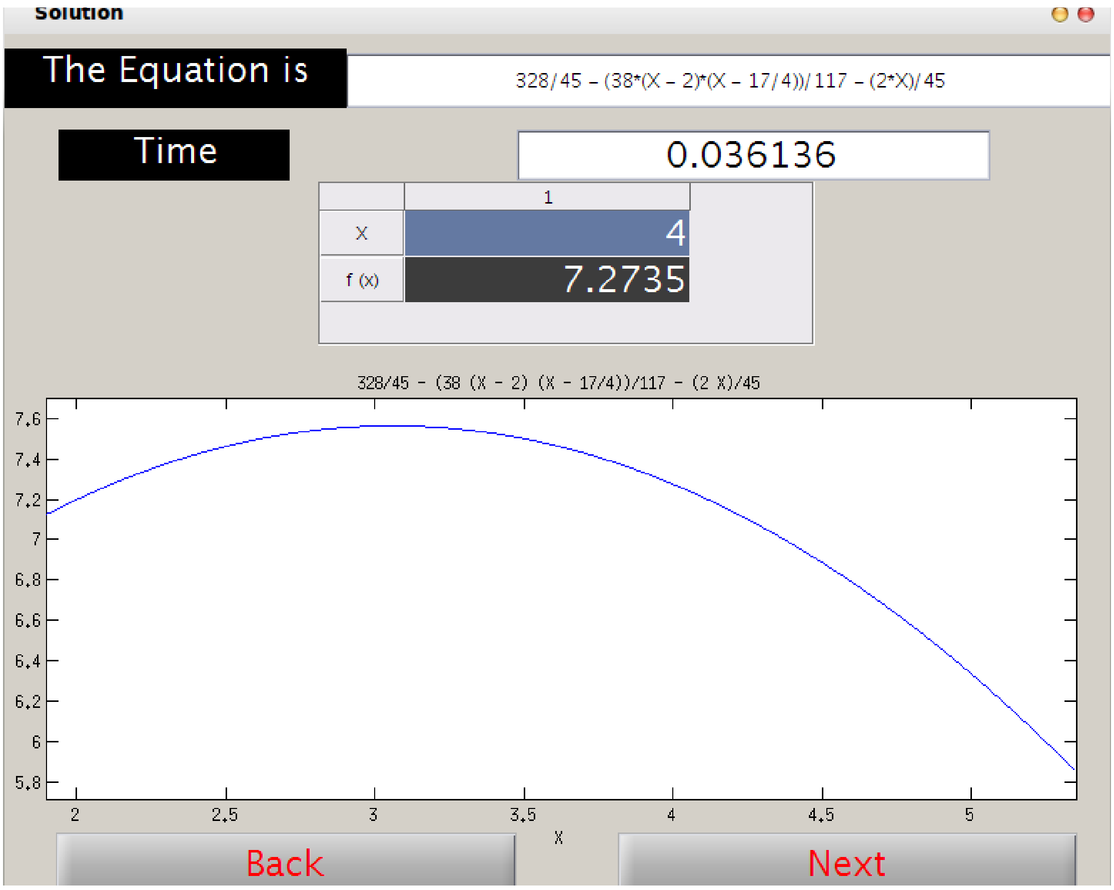
$$x_0 = 2.00, \; y(x_0) = 7.2$$

$$x_1 = 4.25, \; y(x_1) = 7.1$$

$$x_2 = 5.25, \; y(x_2) = 6.0$$




InterpolationData
Enter Order of Equation
2
Enter data points and values
x 2 4.2500 5.2500 7.8100 9.2000 10.6000
f (x) 7.2000 7.1000 6 5 3.5000 5
Add point
Enter points to find values
x 4
Add point
Next
Back to Main Menu


Solution
The Equation is
328/45 – (38*(X – 2)*(X – 17/4))/117 – (2*X)/45
Time
0.036136
x 4
f (x) 7.2735
328/45 - (38 (X - 2) (X - 17/4))/117 - (2 X)/45
X
Back
Next

LAGRANGE INTERPOLATION

1)

The upward velocity of a rocket is given as a function of time in Table 1. Find the velocity at t=16 seconds using the Lagrangian method for **linear** interpolation.

Table 1 Velocity as a function of time

t (s)	$v(t)$ (m/s)
0	0
10	227.04
15	362.78
20	517.35
22.5	602.97
30	901.67

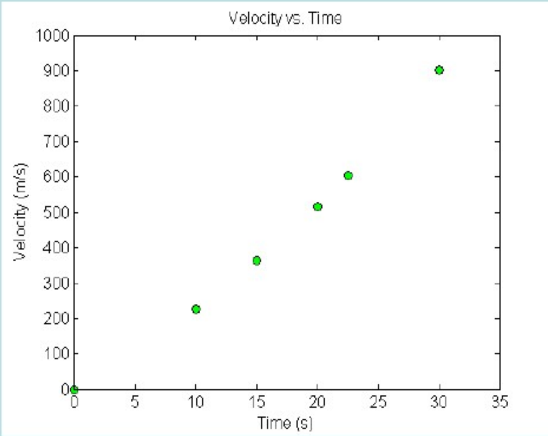
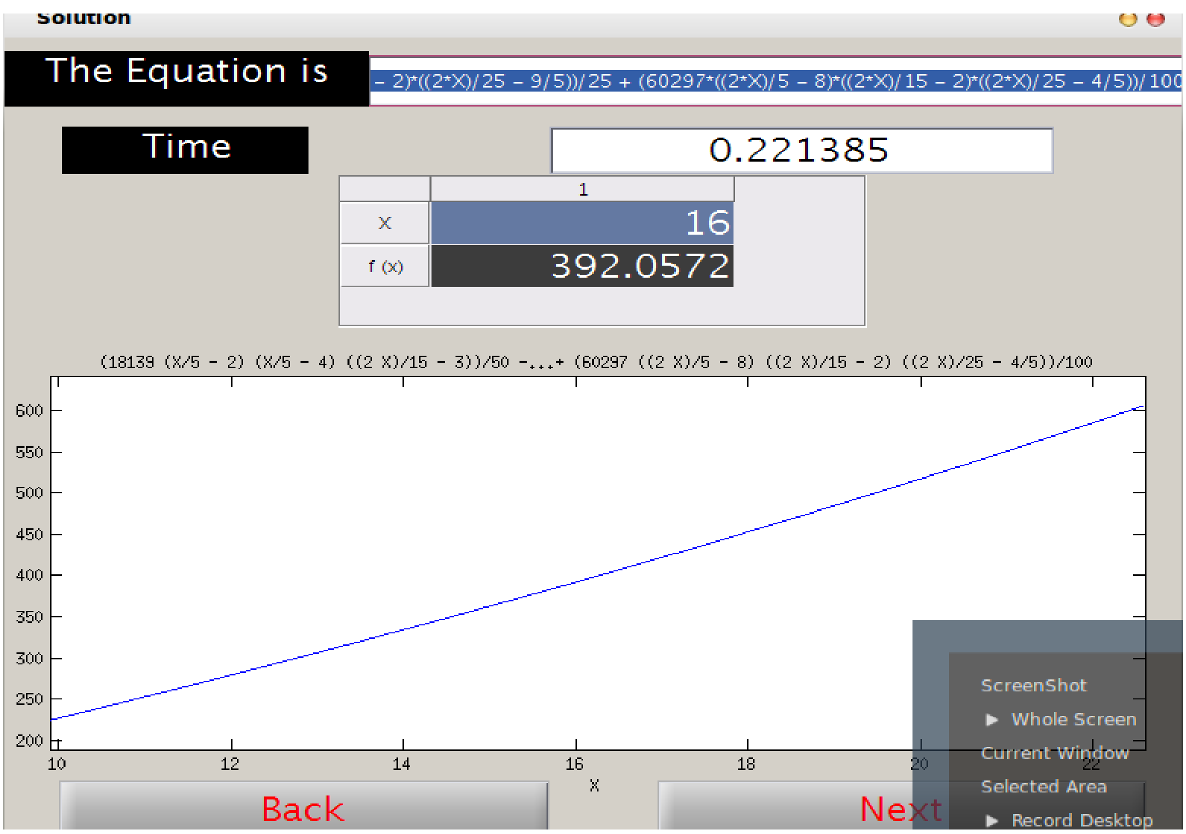
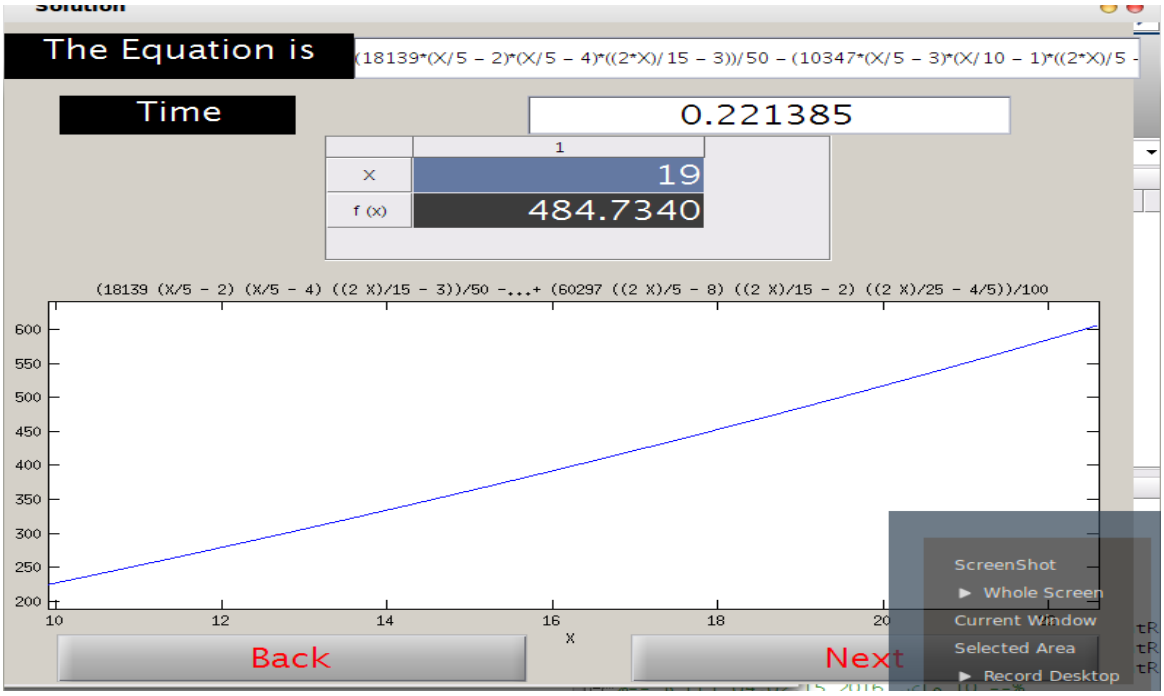


Figure. Velocity vs. time data





2)

Solved Examples

Question 1:

Find the value of y at x = 0 given some set of values (-2, 5), (1, 7), (3, 11), (7, 34)?

Solution:

Given the known values are,

x = 0 ; x_0 = -2 ; x_1 = 1 ; x_2= 3 ; x_3 = 7 ; y_0 = 5 ; y_1 = 7 ; y_2 = 11 ; y_3 = 34

Using the interpolation formula,

$$y = \frac{(x-x_1)(x-x_2).....(x-x_n)}{(x_0-x_1)(x_0-x_2).....(x_0-x_n)}\, y_0 + \frac{(x-x_0)(x-x_2).....(x-x_n)}{(x_1-x_0)(x_1-x_2).....(x_1-x_n)}\, y_1 + ... + \frac{(x-x_1)(x-x_1).....(x-x_{n-1})}{(x_n-x_0)(x_0-x_1).....(x_n-x_{n-1})}\, y_n$$

$$y = \frac{(0-1)(0-3)(0-7)}{(-2-1)(-2-3)(-2-7)} \times 5 + \frac{(0+2)(0-3)(0-7)}{(1+2)(1-3)(1-7)} \times 7 + \frac{(0+2)(0-1)(0-7)}{(3+2)(3-1)(3-7)} \times 11 + \frac{(0+2)(0-1)(0-3)}{(7+2)(7-1)(7-3)} \times 34$$

$$y = \frac{21}{27} + \frac{49}{6} + \frac{-77}{20} + \frac{51}{54}$$

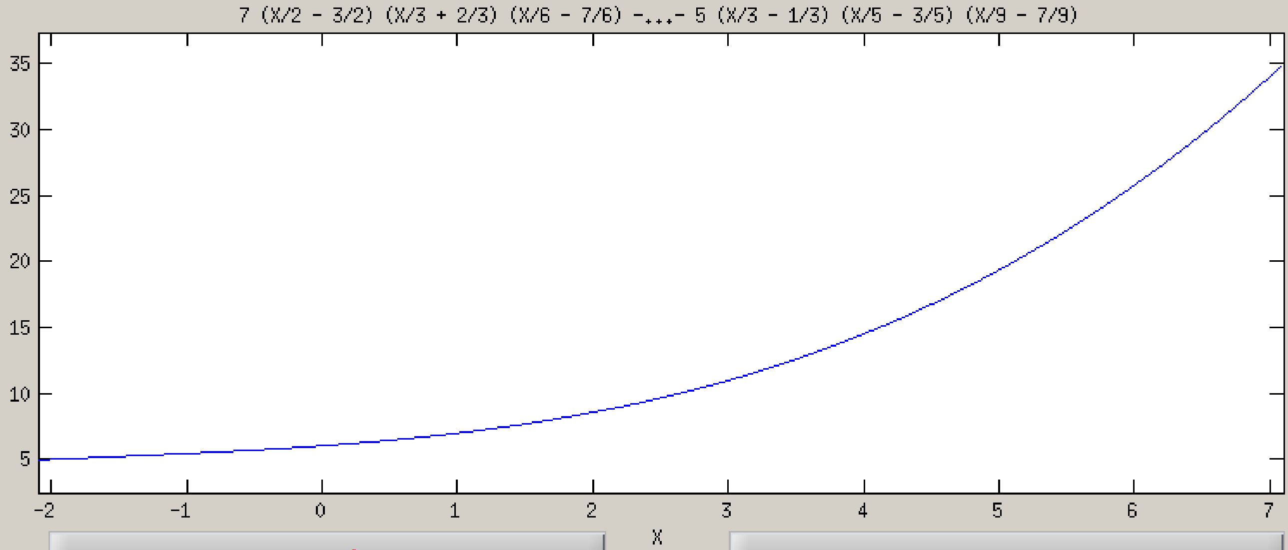
$$y = \frac{1087}{180}$$

Solution

The Equation is /4) + 34*(X/4 – 3/4)*(X/6 – 1/6)*(X/9 + 2/9) – 5*(X/3 – 1/3)*(X/5 – 3/5)*(X/9 – 7/9)

Time 0.050348

	1
X	0
f (x)	6.0389

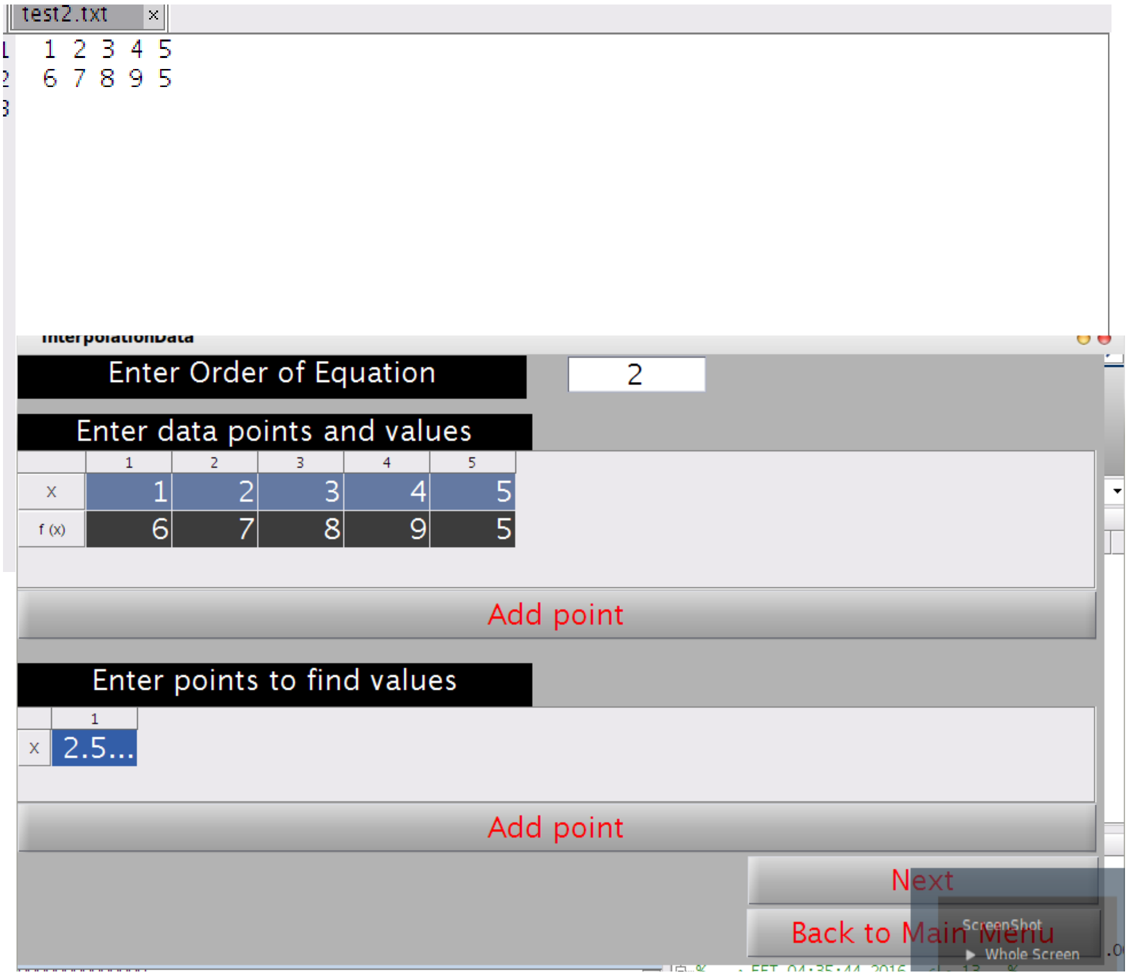


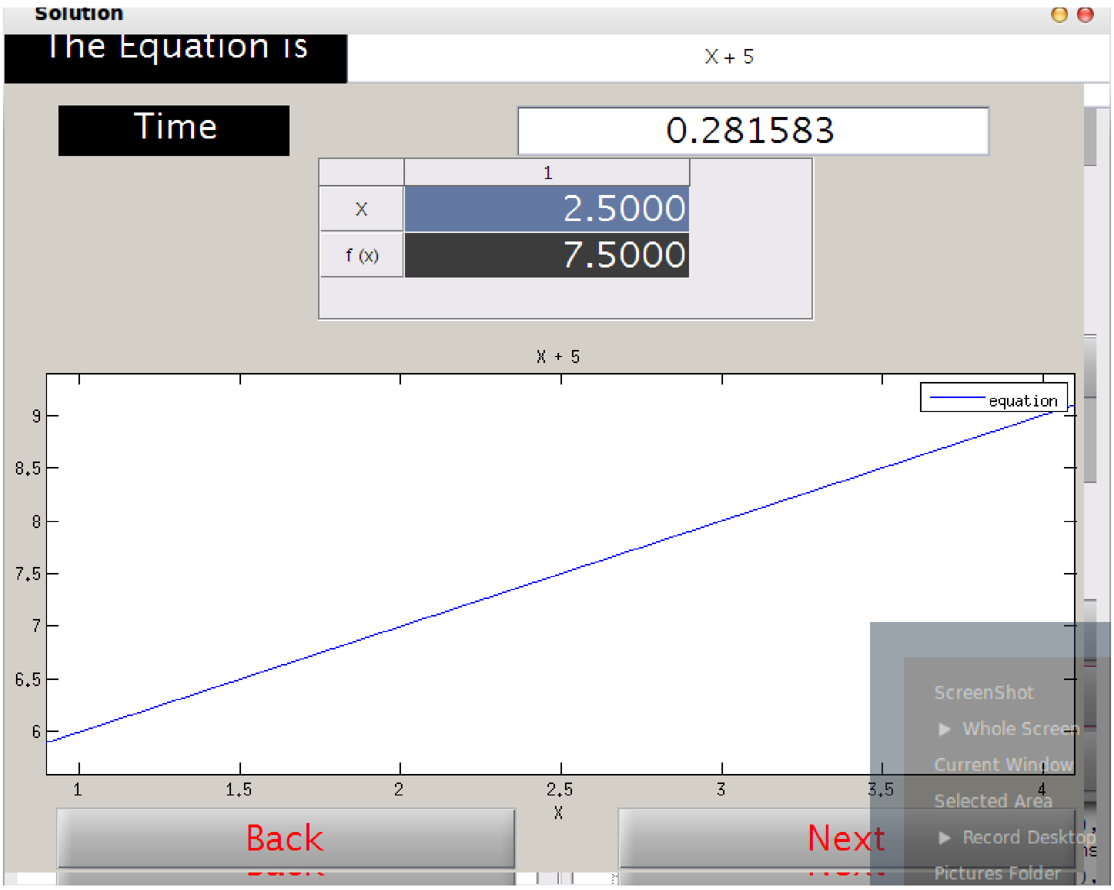
Back

Next

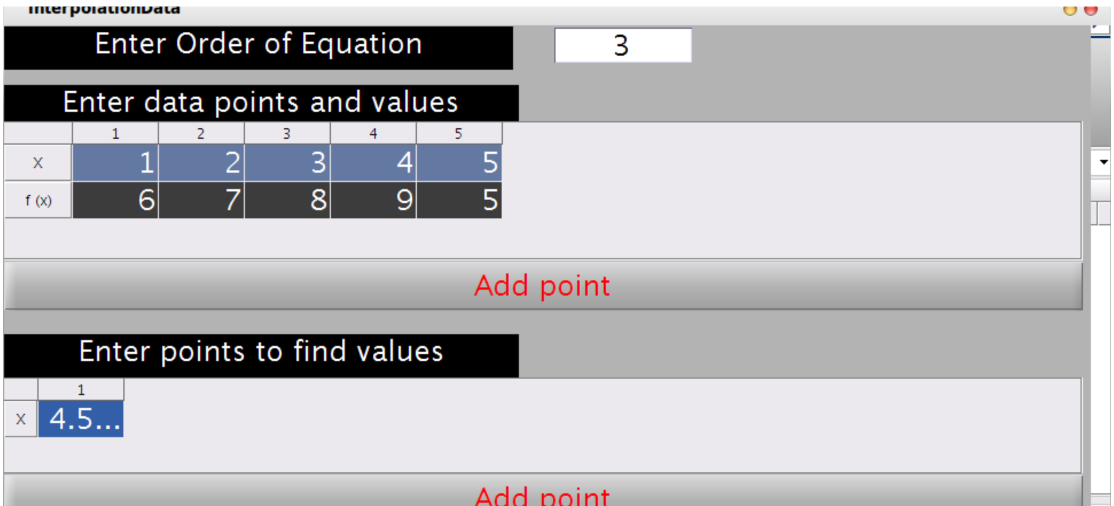
READ FROM FILE

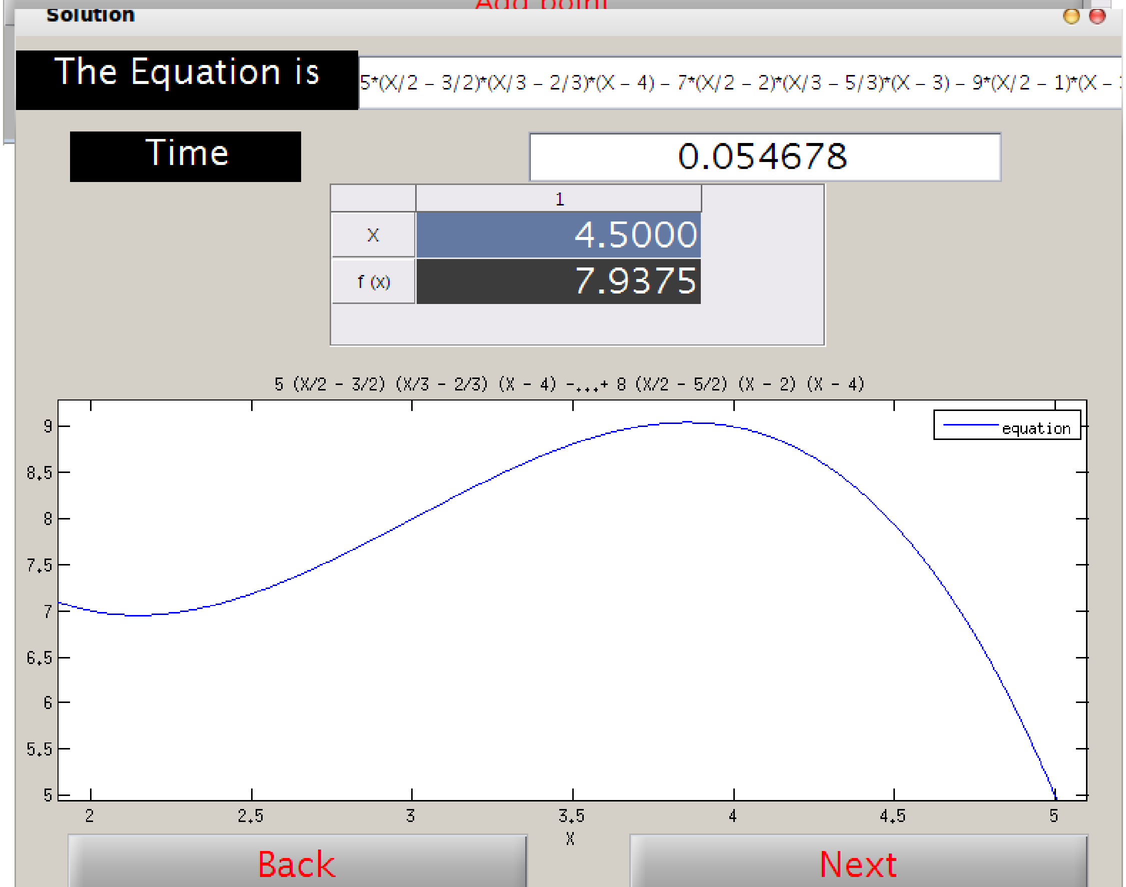
NEWTON INTERPOLATION






test2.txt
1 2 3 4 5
6 7 8 9 5


Enter Order of Equation
3
Enter data points and values
1 2 3 4 5
x 1 2 3 4 5
f (x) 6 7 8 9 5
Add point
Enter points to find values
1
x 4.5...
Add point


Solution
The Equation is
5*(X/2 – 3/2)*(X/3 – 2/3)*(X – 4) – 7*(X/2 – 2)*(X/3 – 5/3)*(X – 3) – 9*(X/2 – 1)*(X –
Time
0.054678
1
x 4.5000
f (x) 7.9375
5 (X/2 - 3/2) (X/3 - 2/3) (X - 4) -...+ 8 (X/2 - 5/2) (X - 2) (X - 4)
equation
9
8.5
8
7.5
7
6.5
6
5.5
5
2
2.5
3
3.5
4
4.5
5
X
Back
Next

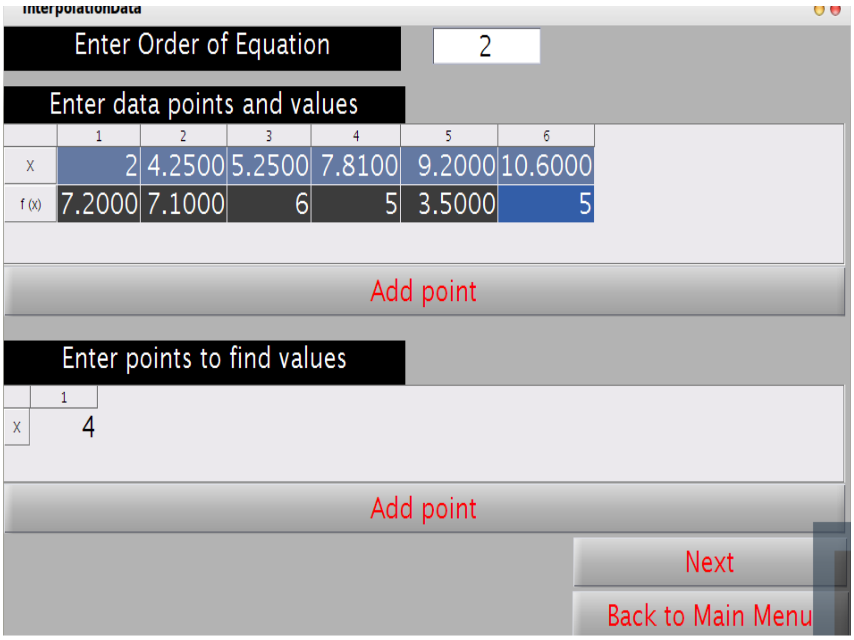
SIMPLE USER GUIDE

- When the user run the project , the main menu will appear and he can choose the part 1 or 2.

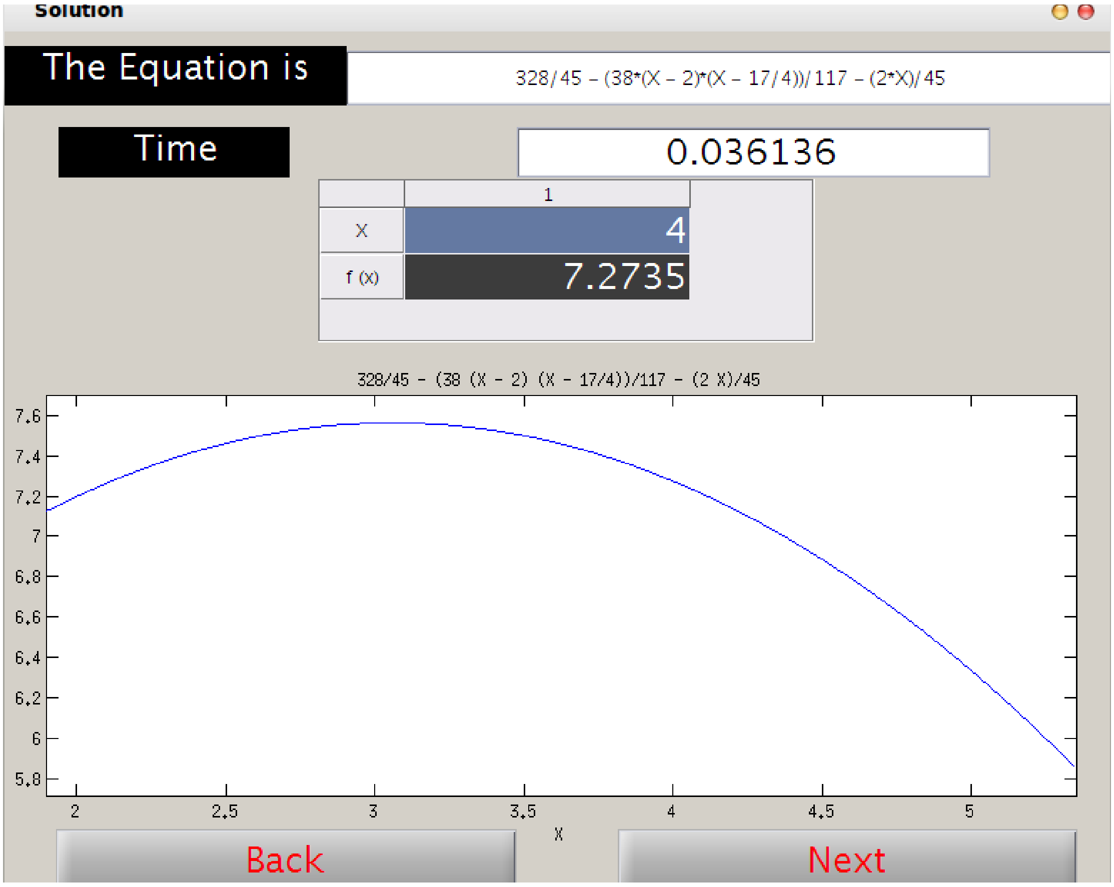
- In part 1: Root Finder
    1. Choose if you want to read the function and variable from file or enter it manually.
    2. Choose the method to solve or press "Next" if you want the general algorithm
    3. Enter the required data then press "Next".
    4. The results will appear and if you want simulation, press"Simulate".
- In Part 2: Interpolation
    1. Choose the method to solve with and press "Next".
    2. Choose if you want to read the given points from file or enter it manually.
    3. Enter the required data and press "Next".

GENERAL ANALYSIS

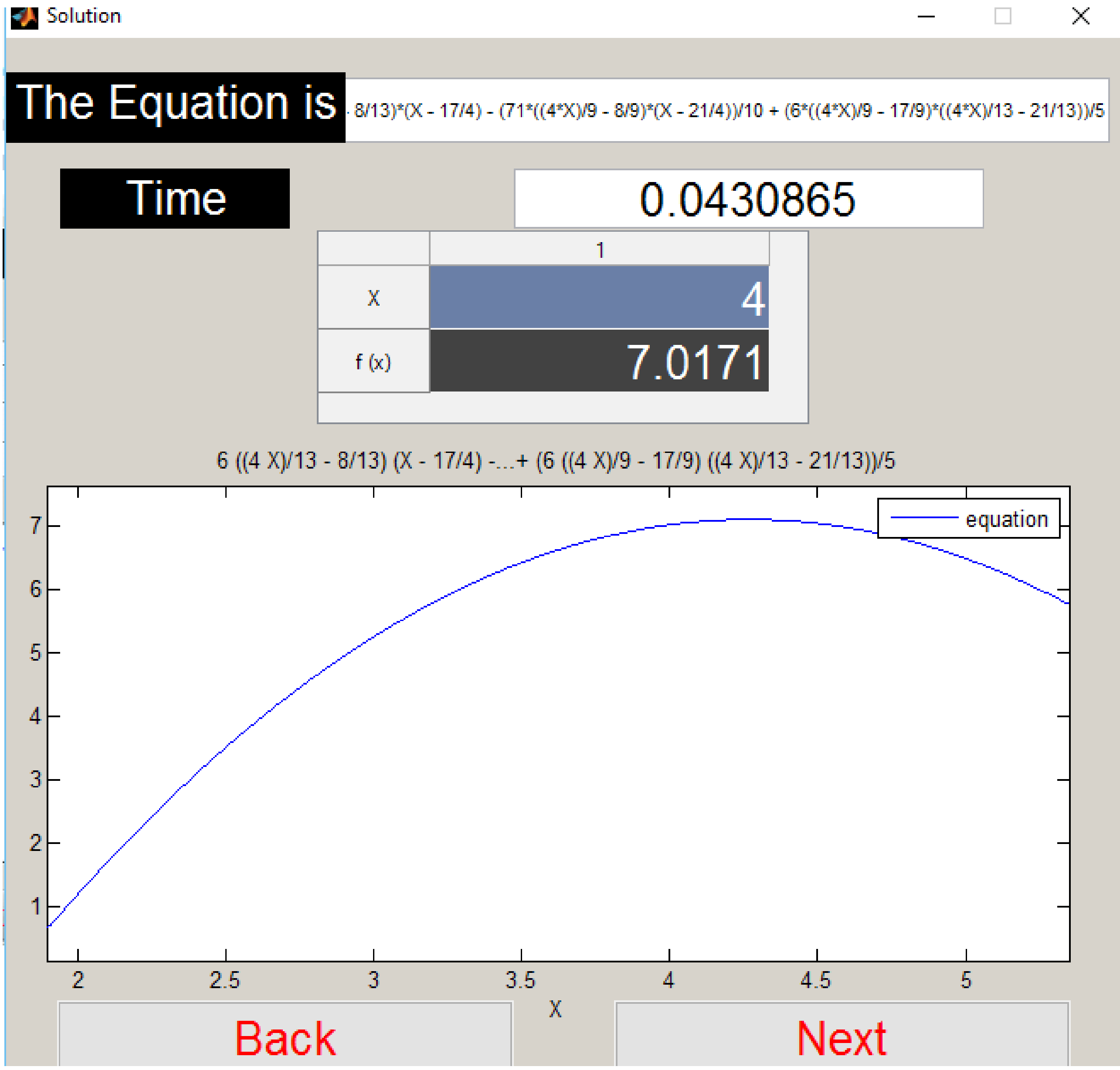
Example 1 :



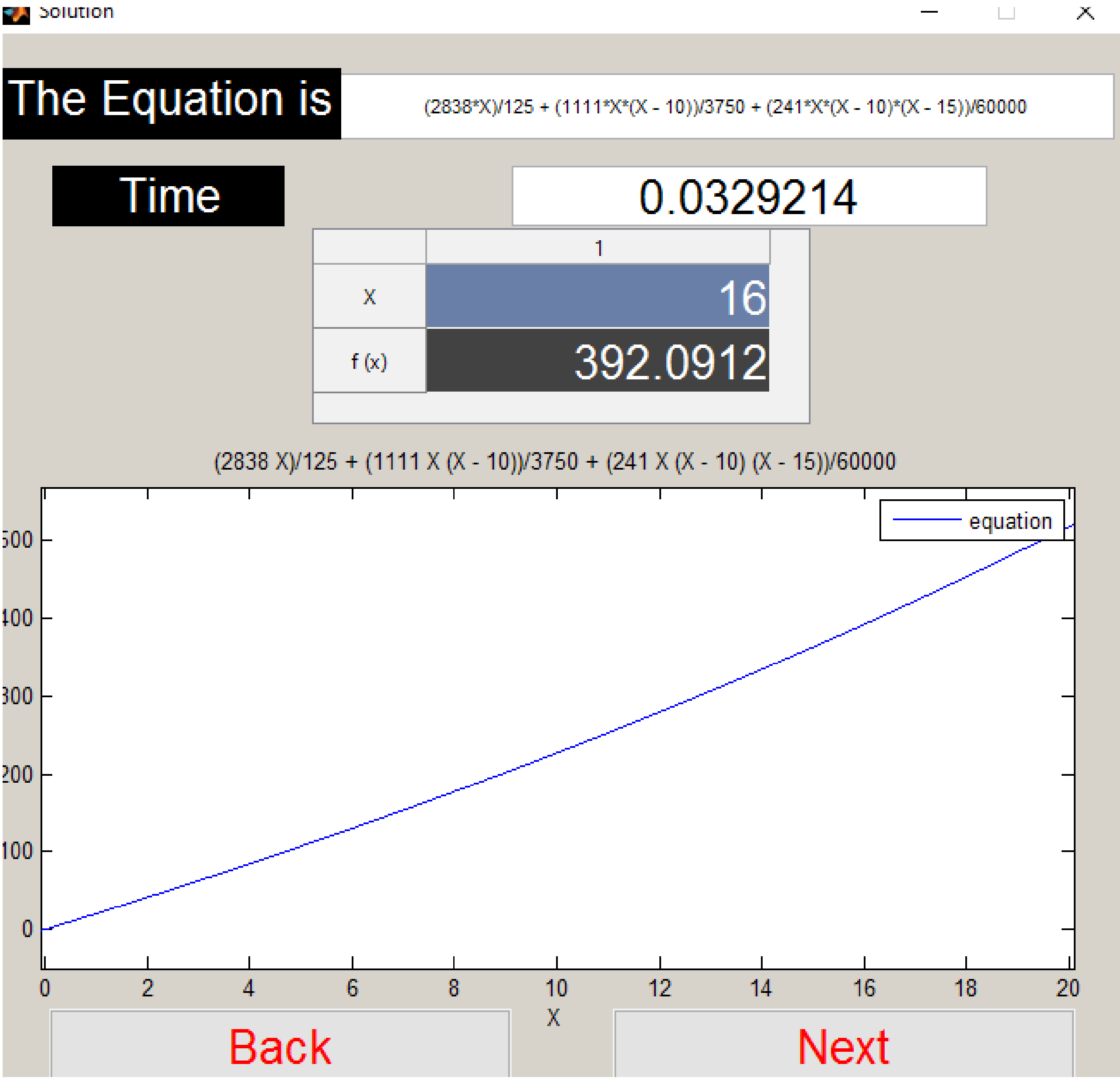
Newton



lagrange


Solution
The Equation is
8/13)*(X - 17/4) - (71*((4*X)/9 - 8/9)*(X - 21/4))/10 + (6*((4*X)/9 - 17/9)*((4*X)/13 - 21/13))/5
Time
0.0430865
1
X
4
f (x)
7.0171
6 ((4 X)/13 - 8/13) (X - 17/4) -...+ (6 ((4 X)/9 - 17/9) ((4 X)/13 - 21/13))/5
equation
X
Back
Next

Example 2: Newton ,, input and lagrange in sample run 1



example 3 : newton,, lagrange in problematic and analysis