**MAPUA UNIVERSITY**

School of Electrical, Electronics

and Computer Engineering

Muralla St. Intramuros, Manila, 1002 Metro Manila

**NUMERICAL METHODS**

**COE60/B1**

**MACHINE PROBLEMS**

**PROJECT**

Submitted by:

Ang, Von Errol

**Submitted to:**

**Janette Fausto**

**Instructor**

**March 16, 2018**

**Table of Contents**

**Discussion**

Machine Problem 1 ………………………………………………………………………………… 3  
Machine Problem 2 ………………………………………………………………………………… 3  
Machine Problem 3 ………………………………………………………………………………… 4  
Machine Problem 4 ………………………………………………………………………………… 6

**User Manual**

Machine Problem 1 ………………………………………………………………………………… 9  
Machine Problem 2 ………………………………………………………………………………… 11  
Machine Problem 3 ………………………………………………………………………………… 13  
Machine Problem 4 ………………………………………………………………………………… 19

**Appendix**

Machine Problem 1 ………………………………………………………………………………… 26  
Machine Problem 2 ………………………………………………………………………………… 31  
Machine Problem 3 ………………………………………………………………………………… 35  
Machine Problem 4 ………………………………………………………………………………… 46

**Reference**

**Discussion**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Machine Problem 1:

**Bracketing Technique (Bisection Method)**

The bisection method (also known as Interval-Halving method) is a root-finding method which repeatedly bisects an interval and then selects a subinterval in which a root must lie for further processing. It is a very simple and robust method, but it is also relatively 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. The method is also called the binary search method and is similar to the Binary Search algorithm in computing, where the range of possible solutions is halved each iteration.

The method is applicable when we wish to solve the equation f(x) = 0 for the real variable x, where f is a continuous function defined on an interval [a, b] and f(a) and f(b) have opposite signs. In this case a and b are said to bracket a root since, by the intermediate value theorem, the f must have at least one root in the 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 off 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.

Machine Problem 2:

**Open Method (Secant Method)**

Although the Bisection Method always converges, the speed of converges is usually too slow for general use. Secant Method is an improvement of the Bisection Method and is also applicable to both polynomials and transcendental equation. First is we initiating values for xo, f(xo), x1 and f(x1). Then solve for x2 which is

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

To continue the process, disregard xo and substitute the previous x1 to xo then substitute the computed x2 to x1. Recomputed for a new root estimate value x2. The process is repeated until the terminating condition is satisfied. The Secant Method does not have the bracketing property of the bisection method. Therefore, the method does not always converge, but when it does converge, it generally does so much faster than the bisection method.

Machine Problem 3:

**Polynomial Technique (Muller’s method)**

Unlike linear systems of equations, nonlinear systems do not have an analytical solution and must be solved using numerical methods. Muller's method is a numerical generalization of the secant method to find roots of nonlinear equations knowing only three points and no information about the derivative of the function. Muller's method is best suited for solving polynomial equations.

* 1. Begin by finding three initial approximations, x0, x1 and x2, of the root of the function f. Muller's method works best when |f(x0)| > |f(x1)| > |f(x2)|. The expression xn denotes x subscript n.
  2. Calculate the coefficients for the quadratic, ax^2 + bx +c, passing through the points:

1. (xn-2 - xn-1, f (xn-2))
2. (0, f (xn-1))
3. xn - xn-1, f(xn))

Where xn-2, xn-1 and xn are the three most recent approximations of the solution x.

* 1. Determine the smallest root of the quadratic ax^2 + bx + c. Use this as the next estimate of x.
  2. Iterate the above process until:

a. The iterative step in x -- that is |xn+1 - xn| -- and the value of the function f evaluation at xn+1 are both sufficiently small;

b. The estimated function fails to have a solution, or;

c. You have iterated some predetermine maximum number of times without meeting condition 1.

If you ended with condition 1, xn+1 is the best approximation of the solution. If you ended with conditions 2 or 3, the function may not have a solution or another method may be better suited.

**Direct Methods (Crout's /Cholesky’s Method)**

Crout's method decomposes a nonsingular *n xn* matrix **A** into the product of an *n×n* lower triangular matrix **L** and an *n×n* unit upper triangular matrix **U**. A unit triangular matrix is a triangular matrix with 1's along the diagonal. Crout's algorithm proceeds as follows:

Evaluate the following pair of expressions for *k* = 0*, . . . , n-*1;

*L*ik = ( *A*ik - (*L*i0 *U*0k +..... + *L*i,k-1 *U*k-1,k)),   for *i = k, . . . , n-*1,

and

*U*kj = *A*kj - (*L*k0 *U*0j + ..... + *L*k,k-1 *U*k-1,j) / *L*kk,   for *j = k+*1*, . . . , n-*1.

Note that *L*ik = 0 for *k > i*, *U*ik = 0 for *k < i*, and *U*kk = 1 for *k =* 0*, … , n-*1. The matrix U forms a unit upper triangular matrix, and the matrix L forms a lower triangular matrix. The matrix A = LU.

After the LU decomposition of A is performed, the solution to the system of linear equations A x = L U x = B is solved by solving the system of linear equations L y = B by forward substitution for y, and then solving the system of linear equations U x = y by backward substitution for x.

Crout's LU decomposition with pivoting is similar to the above algorithm except that for each *k* a pivot row is determined and interchanged with row *k*, the algorithm then proceeds as before. Source code is provided for the two different versions of Crout's LU decomposition, one version performs pivoting and the other version does not. If the matrix A is positive definite symmetric or if the matrix is diagonally dominant, then pivoting is not necessary; otherwise the version using pivoting should be used.

**Iterative Techniques (Gauss-Jacobi Method)**

In [numerical linear algebra](http://en.wikipedia.org/wiki/Numerical_linear_algebra), the Jacobi method is an algorithm for determining the solutions of a [system of linear equations](http://en.wikipedia.org/wiki/System_of_linear_equations) with largest absolute values in each row and column dominated by the diagonal element. Each diagonal element is solved for, and an approximate value plugged in. The process is then iterated until it converges. This algorithm is a stripped-down version of the [Jacobi transformation method of matrix diagonalization](http://en.wikipedia.org/wiki/Jacobi_eigenvalue_algorithm). The method is named after [German](http://en.wikipedia.org/wiki/Germany) mathematician [Carl Gustav Jakob Jacobi](http://en.wikipedia.org/wiki/Carl_Gustav_Jakob_Jacobi).

The Steps are:

1. Rearrange the order of equation in a manner that the dominant coefficient for each unknown lies in the main diagonal of the system of linear equation.
2. Obtain the iterative formula for each variable by transposing the dominant variable and dividing the equation by the dominant coefficient.
3. Begin the iteration by substituting the initial values (0’s) to the variables and obtain the new values. The process is repeated by substituting the variables on the nest iteration until the terminating condition is satisfied.

Machine Problem 4:

**Regression Techniques (Linear Regression)**

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is an explanatory variable, and the other is a dependent variable. For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model.

Before attempting to fit a linear model to observed data, a modeler should first determine if there is a relationship between the variables of interest. This does not necessarily imply that one variable *causes* the other, but that there is some significant association between the two variables. A scatterplot can be a helpful tool in determining the strength of the relationship between two variables. If there appears to be no association between the proposed explanatory and dependent variables (i.e., the scatterplot does not indicate any increasing or decreasing trends), then fitting a linear regression model to the data probably will not provide a useful model. A valuable numerical measure of association between two variables is the correlation coefficient, which is a value between -1 and 1 indicating the strength of the association of the observed data for the two variables.

A linear regression line has an equation of the form ***Y = a + bX***, where ***X*** is the explanatory variable and ***Y*** is the dependent variable. The slope of the line is ***b***, and ***a*** is the intercept (the value of ***y*** when ***x*** = 0).

**Interpolation (Newton’s Divided Difference Interpolating Polynomial)**

In the mathematical field of numerical analysis, a Newton polynomial, named after its inventor Isaac Newton, is the interpolation polynomial for a given set of data points in the Newton form. The Newton polynomial is sometimes called Newton's divided differences interpolation polynomial because the coefficients of the polynomial are calculated using divided differences.

Newton's formula is of interest because it is the straightforward and natural differences-version of Taylor's polynomial. Taylor's polynomial tells where a function will go, based on its y value, and its derivatives (its rate of change, and the rate of change of its rate of change, etc.) at one x value. Newton's formula is Taylor's polynomial based on finite differences instead of instantaneous rates of change.

As with other difference formulas, the degree of a Newton's interpolating polynomial can be increased by adding more terms and points without discarding existing ones. Newton's form has the simplicity that the new points are always added at one end: Newton's forward formula can add new points to the right, and Newton's backwards formula can add new points to the left. Unfortunately, the accuracy of polynomial interpolation depends on how close the interpolated point is to the middle of the x values of the set of points used; as Newton's form always adds new points at the same end, an increase in degree cannot be used to increase the accuracy anywhere but at that end. Gauss, Stirling, and Bessel all developed formulae to remedy that problem.

Gauss's formula alternately adds new points at the left and right ends, thereby keeping the set of points centered near the same place (near the evaluated point). When so doing, it uses terms from Newton's formula, with data points and x values renamed in keeping with one's choice of what data point is designated as the x0 data point.

Stirling's formula remains centered about a data point, for use when the evaluated point is nearer to a data point than to a middle of two data points. Bessel's formula remains centered about a middle between two data points, for use when the evaluated point is nearer to a middle than to a data point. They achieve that by sometimes using the average of two differences where Newton's or Gauss's would use just one difference. Stirling's does that in odd-degree terms; Bessel does that in even-degree terms. Calculating and averaging two differences need not involve extra work, since it can be done by formula, in advance—the expression for the averaged difference is not more complicated than that of the simple difference.

**Numerical Integration (Trapezoidal Rule)**

In numerical analysis, numerical integration constitutes a broad family of algorithms for calculating the numerical value of a definite integral, and by extension, the term is also sometimes used to describe the numerical solution of differential equations. This article focuses on calculation of definite integrals. The term numerical quadrature (often abbreviated to quadrature) is a synonym for numerical integration, especially as applied to one-dimensional integrals. Numerical integration over more than one dimension is sometimes incorrectly described as cubature, since the meaning of quadrature is understood for higher-dimensional integration as well. In numerical analysis, the trapezoidal rule (also known as the trapezoid rule or trapezium rule) is a technique for approximating the definite integral

 \int_{a}^{b} f(x)\,dx.

The trapezoidal rule works by approximating the region under the graph of the function f(x) as a trapezoid and calculating its area. It follows that

 \int_{a}^{b} f(x)\, dx \approx (b-a) \left[\frac{f(a) + f(b)}{2} \right].

**Numerical Differentiation (Forward/Backward/Centered Finite Divided Difference)**

A finite difference is a mathematical expression of the form f(x + b) − f(x + a). If a finite difference is divided by b − a, one gets a difference quotient. The approximation of derivatives by finite differences plays a central role in finite difference methods for the numerical solution of differential equations, especially boundary value problems.

Certain recurrence relations can be written as difference equations by replacing iteration notation with finite differences.

Today, the term "finite difference" is often taken as synonymous with finite difference approximations of derivatives, especially in the context of numerical methods. Finite difference approximations are finite difference quotients in the terminology employed above.

Three forms are commonly considered: forward, backward, and central differences.

A **forward difference** is an expression of the form

![ \Delta_h[f](x) =  f(x + h) - f(x). \ ](data:image/png;base64,)

Depending on the application, the spacing *h* may be variable or constant. When omitted, *h* is taken to be 1: ![\Delta[f](x) = \Delta_1[f](x)](data:image/png;base64,).

A **backward difference** uses the function values at *x* and *x* − *h*, instead of the values at *x* + *h* and *x*:

![ \nabla_h[f](x) =  f(x) - f(x-h). \ ](data:image/png;base64,)

Finally, the **central difference** is given by

![ \delta_h[f](x) =  f(x+\tfrac12h)-f(x-\tfrac12h). \ ](data:image/png;base64,)

**USER MANUAL\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

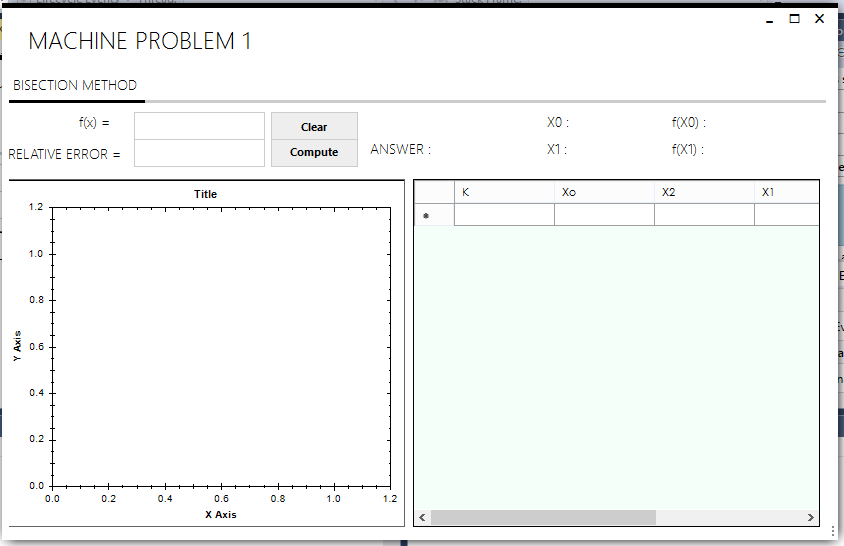
Machine Problem 1:

**Execution of the program:**

1. Open Program folder
2. Double click “MP1.exe” to open the application.

**Proper use of the program:**

**BISECTION TAB**

Fig.1: Bisection Method Tab

Steps:

1. Enter the function to be solved in terms of x and the relative error.
2. Click Compute

After clicking compute, the program would show the answer/root and the iterations undergone and the graph of the function.

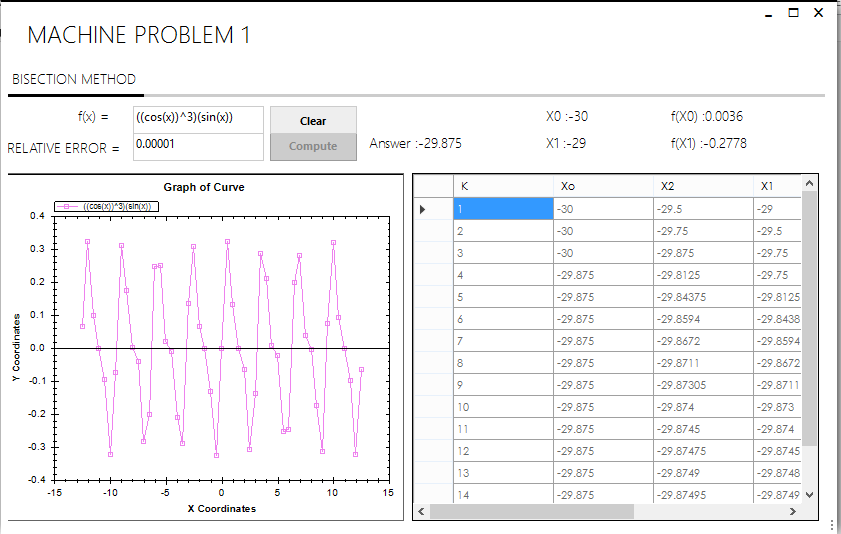


Fig.2: Bisection Method Sample Results

Machine Problem 2:

**Execution of the program:**

1. Open Program folder
2. Double click “MP2.exe” to open the application.

**Proper use of the program:**

**SECANT TAB**

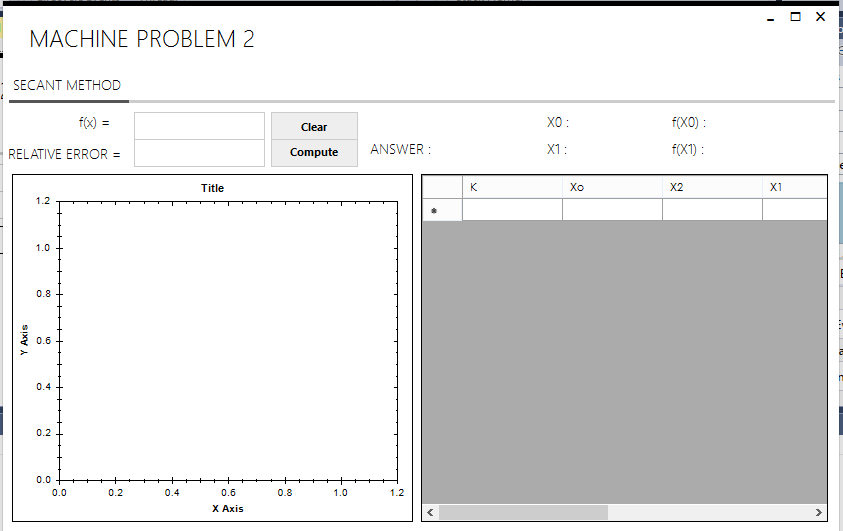


Fig.1: Secant Method Tab

Steps:

1. Enter the function to be solved in terms of x and the relative error.
2. Click Compute

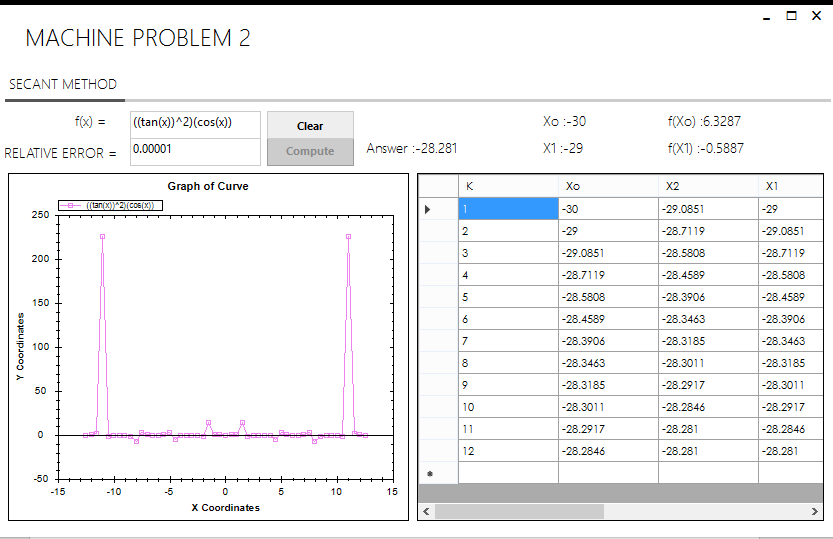
After clicking compute, the program would show the answer/root and also the iterations undergone and the graph of the function. 

Fig.2: Secant Method Sample Results

Machine Problem 3:

**Execution of the program:**

1. Open Program folder
2. Double click “MP3.exe” to open the application.

**Proper use of the program:**

The program consists of different tabs which contains different techniques or methods. (See Fig.1)

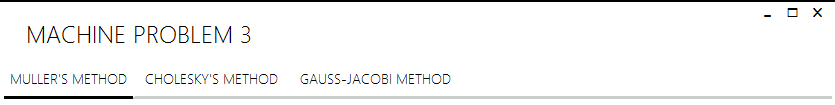


Fig.1: The tabs in the program

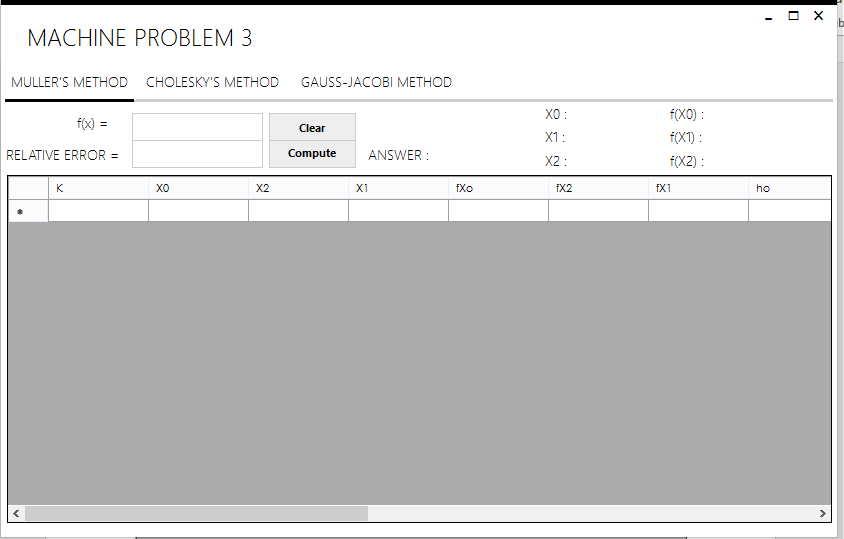
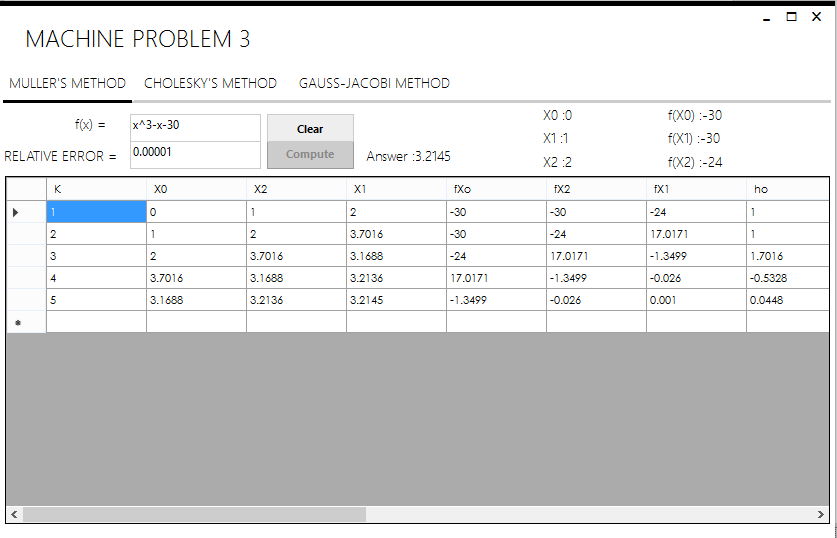
**Muller’s Method Tab**

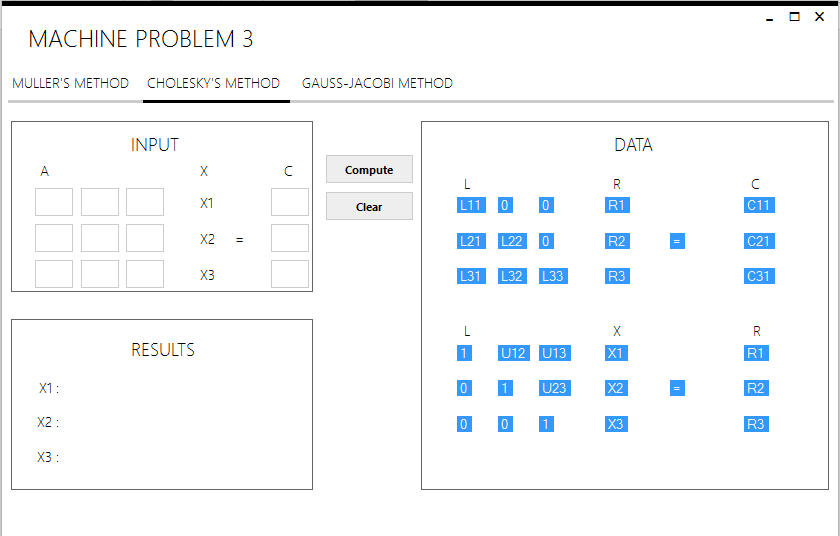
Fig.2: Muller’s Method Tab

Steps:

1. Enter the function to be solved in terms of x and the absolute error.
2. Click “Compute”

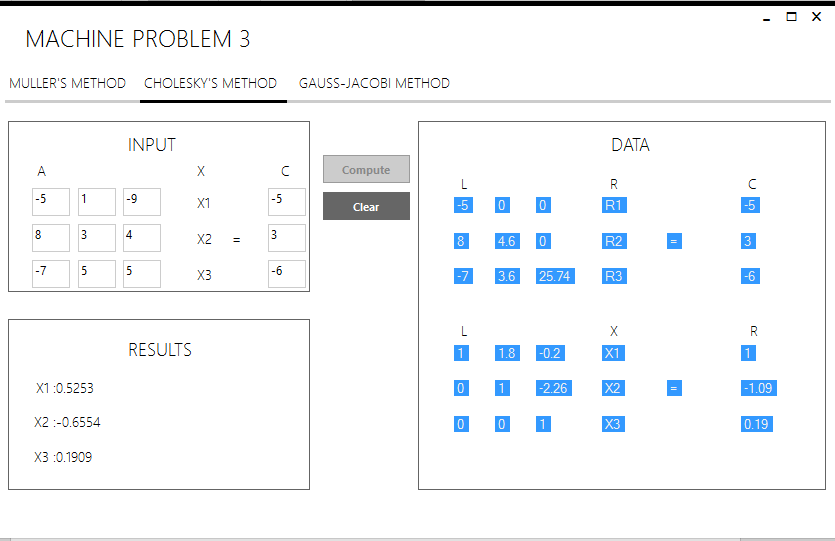
Fig.3: Muller’s Method Sample Results

**Cholesky’s Method Tab**

Fig.4: Cholesky’s Method Tab

Steps:

1. Enter the Matrix A and C.
2. Click “Compute” button.

Fig.5: Muller’s Method Tab

**Gauss-Jacobi Method TAB**

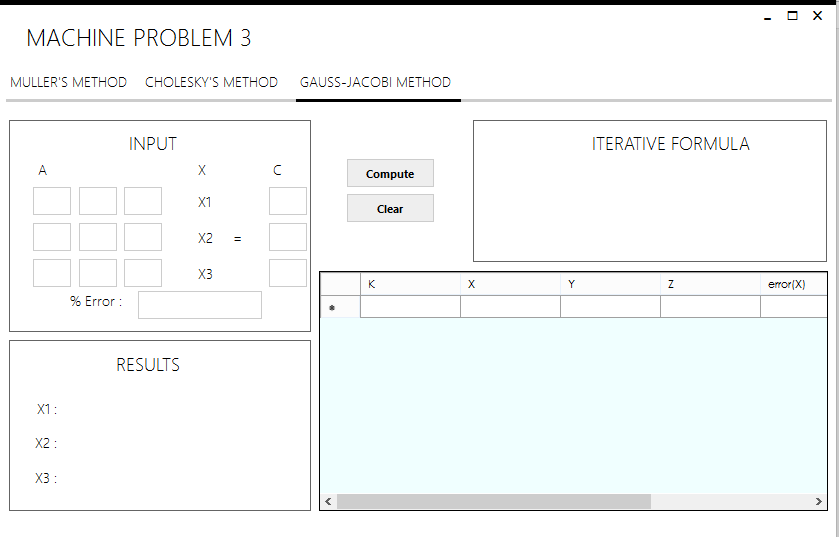
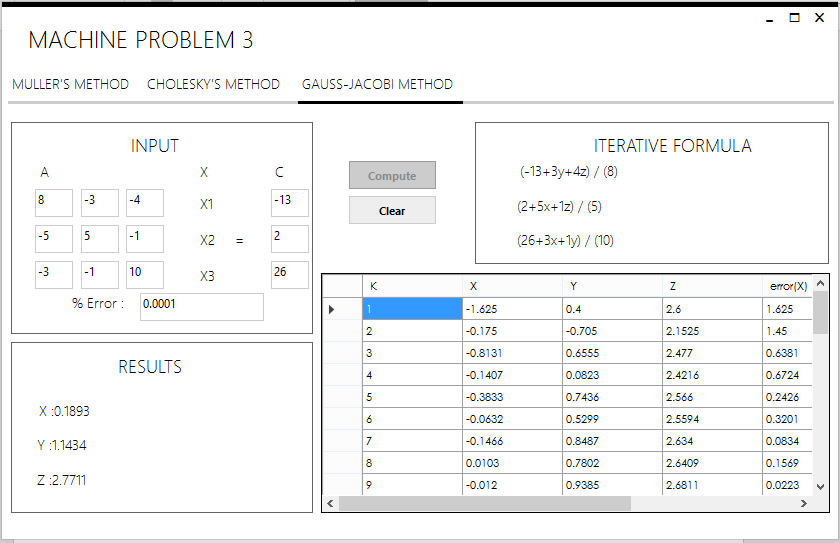


Fig.6: Gauss-Jacobi Method Tab

Steps:

1. Enter the Matrix A and C and % Error.
2. Click “Compute” button.

Fig.7: Gauss-Jacobi Method Sample Results

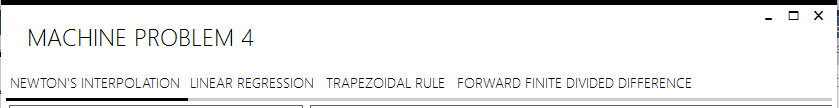
Machine Problem 4:

**Execution of the program:**

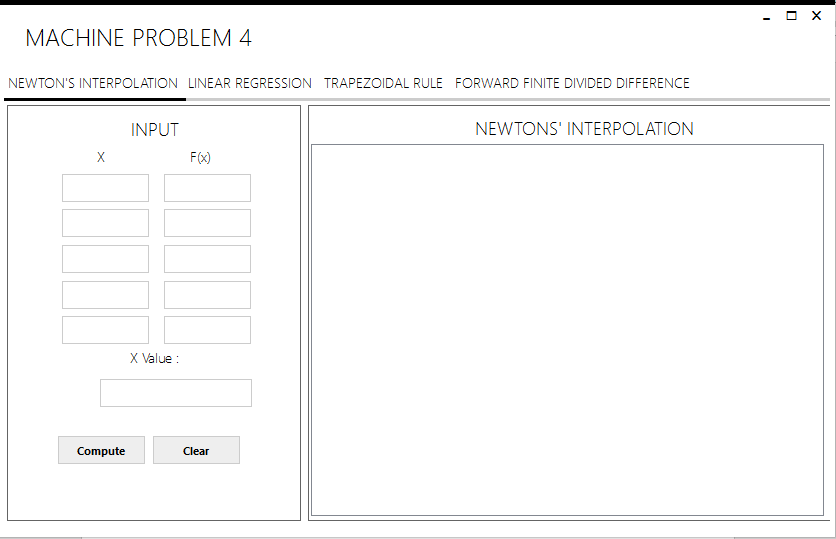
1. Open Program folder.
2. Double click the “MP4.exe” with Application type.

**Proper use of the program:**

The program consists of different tabs which contains different techniques or methods. (See Fig.1)

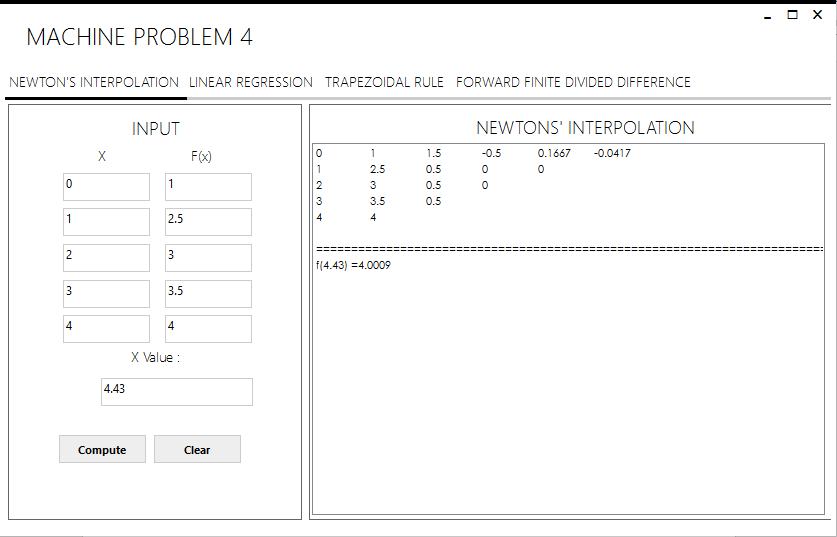
Fig.1: The tabs in the program

**Newton’s Interpolation Tab**

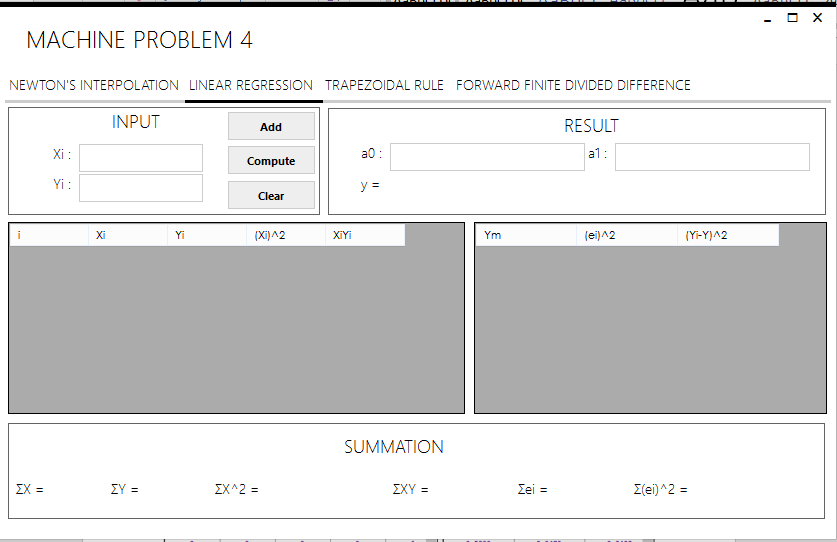
Fig.2: Newton’s Interpolation Tab

Steps:

1. Enter the desired X and F(x) values.
2. Enter the desired Value of X.
3. Click “Compute” button.

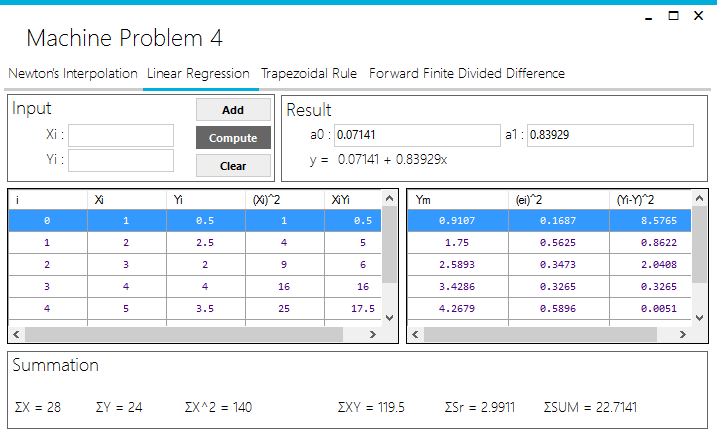
Fig.3: Newton’s Interpolation Sample Result

**Linear Regression Tab**

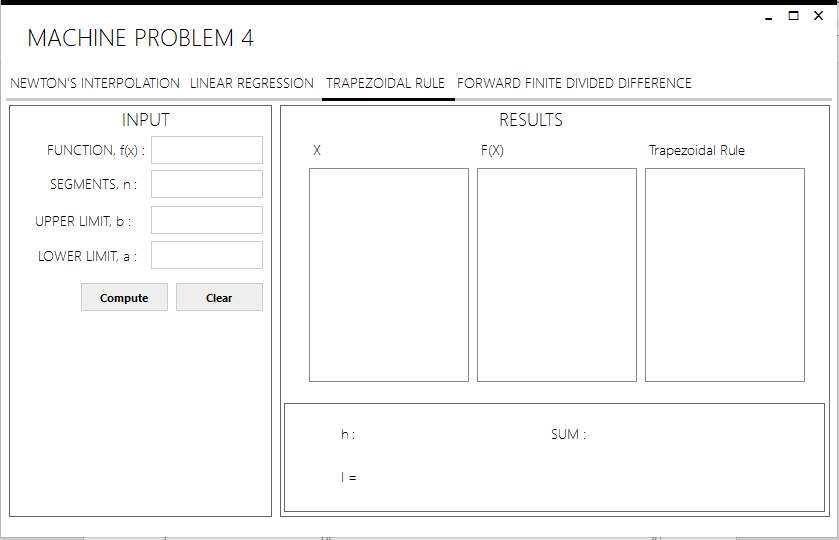
Fig.4: Linear Regression Tab

Steps:

1. Enter the desired Xi and Yi values.
2. Click Add
3. Repeat step1-2 until all data points are added.
4. Click “Compute” button.

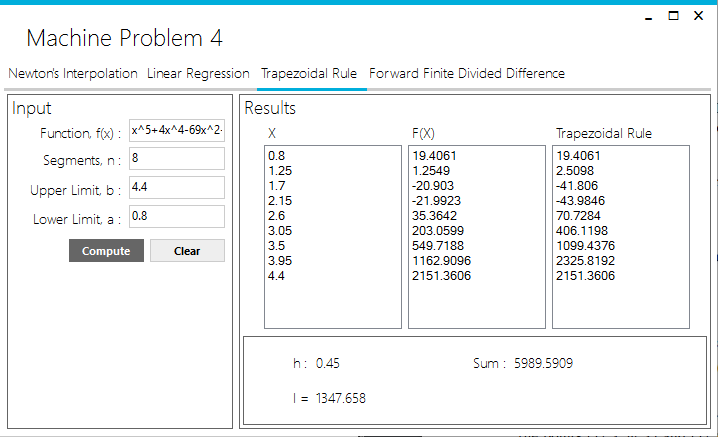
Fig.5: A sample result; It shows the value of A0 and A1 which forms the equation. It also displays the tabulated form of the important values and determine its sum.

**Trapezoidal Rule Tab**

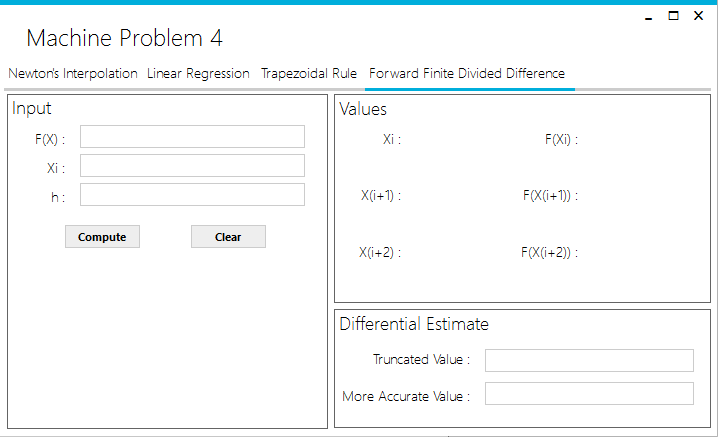
Fig.6: Trapezoidal Rule Tab

Steps:

1. Enter the desired function.
2. Input number of segments.
3. Enter upper and lower limit values.
4. Click “Compute” button.

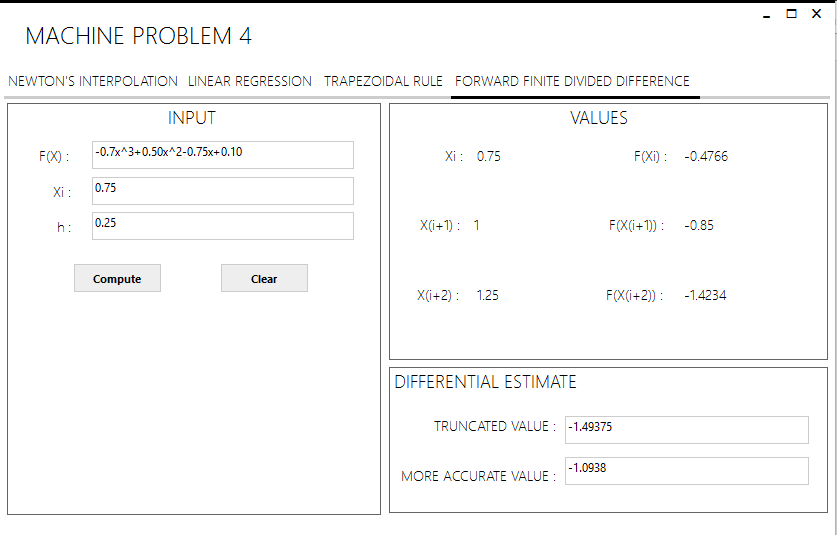
Fig.7: Sample result of Trapezoidal Rule

**Forward Finite Divided Difference Tab**

Fig.8: Forward Finite Divided Difference Tab

Steps:

1. Enter the desired function
2. Input number of segments
3. Enter upper and lower limit values.
4. Click “Compute” button.

Fig.9: Sample result of Forward Finite Divided Difference (FFDD)

**APPENDIX\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Machine Problem 1:

File: Form1.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Windows.Forms;  using System.Collections;  using MetroFramework.Forms;  using info.lundin.math;  using ZedGraph;  namespace MP1  {  public partial class Form1 : MetroForm  {  public Form1()  {  InitializeComponent();  myparse = new ExpressionParser();  myhash = new Hashtable();  table();  }  private double xo, x1, x2 = 0.00000, errornum, fxo, fx1, fx2 = 0.000, prevnum = 0.0000, ans, test;  private void metroLabel1\_Click(object sender, EventArgs e)  {  }  private void Form1\_Load(object sender, EventArgs e)  {  }  double[] col1 = new double[10];  double[] col2 = new double[10];  private void metroButton2\_Click\_1(object sender, EventArgs e)  {  clear(1);  // Command1.Hide();  metroButton2.Visible = false;  metroTextBox1.Clear();  metroTextBox2.Clear();  metroButton1.Enabled = true;  dataGridView1.Visible = true;  labelanswerBM.Visible = false;  zedGraphControl1.Visible = true;  }  double[] col3 = new double[10];  private ExpressionParser myparse;  private Hashtable myhash;  private void metroButton1\_Click(object sender, EventArgs e)  {  dataGridView1.Visible = true;  labelanswerBM.Visible = true;  metroButton2.Visible = true;  metroButton1.Enabled = false;  int j, x = 1;  try  {  clear(1);  plot\_graph(zedGraphControl1, x);  errornum = double.Parse(metroTextBox2.Text);  get\_num(x);  labelx0.Text = labelx0.Text + xo.ToString();  labelx1.Text = labelx1.Text + x1.ToString();  labelfx0.Text = labelfx0.Text + fxo.ToString();  labelfx1.Text = labelfx1.Text + fx1.ToString();  for (j = 0; ; j++)  {  test = Math.Round(Math.Abs(x2 - prevnum), 4);  prevnum = Math.Round(x2, 4);  x2 = Math.Round((x1 + xo) / 2, 5);  myhash.Clear();  myhash.Add("x", x2.ToString());  fx2 = Math.Round(myparse.Parse(metroTextBox1.Text, myhash), 4);  dataGridView1.Rows.Add();  dataGridView1.Rows[j].Cells[0].Value = j + 1;  dataGridView1.Rows[j].Cells[1].Value = xo;  dataGridView1.Rows[j].Cells[2].Value = x2;  dataGridView1.Rows[j].Cells[3].Value = x1;  dataGridView1.Rows[j].Cells[4].Value = fx2;  dataGridView1.Rows[j].Cells[5].Value = test;  if (fx2 < 0)  {  if (fxo < 0)  {  xo = Math.Round(x2, 4);  fxo = Math.Round(fx2, 4);  }  else  {  x1 = Math.Round(x2, 4);  fx1 = Math.Round(fx2, 4);  }  }  else  {  if (fxo > 0)  {  xo = Math.Round(x2, 4);  fxo = Math.Round(fx2, 4);  }  else  {  x1 = Math.Round(x2, 4);  fx1 = Math.Round(fx2, 4);  }  }  if (j != 0)  if (Math.Round(Math.Abs(x2 - prevnum), 4) <= errornum)  {  {  ans = Math.Round(x2, 4);  test = Math.Round(Math.Abs(x2 - prevnum), 4);  dataGridView1.Rows.Add();  dataGridView1.Rows[j + 1].Cells[0].Value = j + 2;  dataGridView1.Rows[j + 1].Cells[1].Value = xo;  dataGridView1.Rows[j + 1].Cells[2].Value = x2;  dataGridView1.Rows[j + 1].Cells[3].Value = x1;  dataGridView1.Rows[j + 1].Cells[4].Value = fx2;  dataGridView1.Rows[j + 1].Cells[5].Value = test;  break;  }  }  }  labelanswerBM.Text = labelanswerBM.Text + ans.ToString("#.####");  }  catch (Exception a)  {  MessageBox.Show(a.Message, "Error encountered!", MessageBoxButtons.OK);  }  zedGraphControl1.Visible = true;  }  private void get\_num(int x)  {  for (int i = -30; ; i++)  {  myhash.Clear();  myhash.Add("x", i.ToString());  xo = i;  fxo = Math.Round(myparse.Parse(metroTextBox1.Text, myhash), 4);  myhash.Clear();  myhash.Add("x", (i + 1).ToString());  x1 = i + 1;  fx1 = Math.Round(myparse.Parse(metroTextBox1.Text, myhash), 4);  if ((fx1 < 0 && fxo > 0) || (fx1 > 0 && fxo < 0))  {  break;  }  }  }  public void clear(int v)  {  labelx0.Text = "X0 :";  labelx1.Text = "X1 :";  labelfx0.Text = "f(X0) :";  labelfx1.Text = "f(X1) :";  labelanswerBM.Text = "Answer :";  prevnum = 0.00; test = 0.00; x2 = 0.00; x1 = 0.00; xo = 0.00; fxo = 0.0000; fx1 = 0.00;  ans = 0.00; test = 0.00;// x3 = 0.00; fx3 = 0.00;  zedGraphControl1.GraphPane.CurveList.Clear();;  dataGridView1.Rows.Clear();  dataGridView1.Columns.Clear();  table();  }  public void table()  {  dataGridView1.Columns.Add("K", "K");  dataGridView1.Columns.Add("xo", "Xo");  dataGridView1.Columns.Add("x2", "X2");  dataGridView1.Columns.Add("x1", "X1");  dataGridView1.Columns.Add("fx2", "fX2");  dataGridView1.Columns.Add("error", "Error");  }  private void plot\_graph(ZedGraphControl zgc, int a)  {  // get a reference to the GraphPane  GraphPane myPane = zgc.GraphPane;  // Set the Titles  myPane.Title.Text = "Graph of Curve";  myPane.XAxis.Title.Text = "X Coordinates";  myPane.YAxis.Title.Text = "Y Coordinates";  PointPairList list1 = new PointPairList();  double y = 0;  double x;  for (int i = -25; i < 26; i++)  {  myhash.Clear();  x = (double)i \* 0.5;  myhash.Add("x", x.ToString());  y = myparse.Parse(metroTextBox1.Text, myhash);  list1.Add(x, y);  }  LineItem myCurve = myPane.AddCurve(metroTextBox1.Text, list1, Color.Violet, SymbolType.Square);  zgc.AxisChange();  zgc.Invalidate();  zgc.Refresh();  }  }  } |

Machine Problem 2:

File: Form1.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Windows.Forms;  using System.Collections;  using MetroFramework.Forms;  using info.lundin.math;  using ZedGraph;  namespace MP2  {  public partial class Form1 : MetroForm  {  public Form1()  {  InitializeComponent();  myparse = new ExpressionParser();  myhash = new Hashtable();  table();  myparse.Values.Add("x", 0);  }  private double xo, x1, x2 = 0.00000, errornum, fxo, fx1, fx2 = 0.000, prevnum = 0.0000, ans, test, xn, xn1, dfxn, fxn;  double[] col1 = new double[10];  double[] col2 = new double[10];  double[] col3 = new double[10];  //private int p = 0;  private ExpressionParser myparse;  private Hashtable myhash;  private void metroButton1\_Click(object sender, EventArgs e)  {  metroButton2.Visible = true;  labelanswer.Visible = true;  metroButton1.Enabled = false;  try  {  int x = 2, i = 0;  clear(2);  get\_num(x);  plot\_graph(zedGraphControl1, 2);  labelfx0.Text = labelfx0.Text + fxo.ToString();  labelfx1.Text = labelfx1.Text + fx1.ToString();  labelx0.Text = labelx0.Text + xo.ToString();  labelx1.Text = labelx1.Text + x1.ToString();  errornum = double.Parse(metroTextBox2.Text);  x2 = Math.Round(x1 - (fx1) \* ((x1 - xo) / (fx1 - fxo)), 4);  myparse.Values["x"].SetValue(x2);  fx2 = Math.Round(myparse.Parse(metroTextBox1.Text), 4);  while (Math.Round(Math.Abs(x2 - prevnum), 4) > errornum)  {  test = Math.Round(Math.Abs(x2 - prevnum), 4);  dataGridView1.Rows.Add();  dataGridView1.Rows[i].Cells[0].Value = i + 1;  dataGridView1.Rows[i].Cells[1].Value = xo;  dataGridView1.Rows[i].Cells[2].Value = x2;  dataGridView1.Rows[i].Cells[3].Value = x1;  dataGridView1.Rows[i].Cells[4].Value = fxo;  dataGridView1.Rows[i].Cells[5].Value = fx2;  dataGridView1.Rows[i].Cells[6].Value = fx1;  dataGridView1.Rows[i].Cells[7].Value = test;  if (i != 0)  {  prevnum = x2;  }  xo = Math.Round(x1, 4);  x1 = Math.Round(x2, 4);  fxo = Math.Round(fx1, 4);  fx1 = Math.Round(fx2, 4);  x2 = Math.Round(x1 - (fx1) \* ((x1 - xo) / (fx1 - fxo)), 4);  i++;  myparse.Values["x"].SetValue(x2);  fx2 = Math.Round(myparse.Parse(metroTextBox1.Text), 4);  }  test = Math.Round(Math.Abs(x2 - prevnum), 4);  dataGridView1.Rows.Add();  dataGridView1.Rows[i].Cells[0].Value = i + 1;  dataGridView1.Rows[i].Cells[1].Value = xo;  dataGridView1.Rows[i].Cells[2].Value = x2;  dataGridView1.Rows[i].Cells[3].Value = x1;  dataGridView1.Rows[i].Cells[4].Value = fxo;  dataGridView1.Rows[i].Cells[5].Value = fx2;  dataGridView1.Rows[i].Cells[6].Value = fx1;  dataGridView1.Rows[i].Cells[7].Value = test;  ans = Math.Round(x2, 4);  labelanswer.Text = labelanswer.Text + ans.ToString();  }  catch (Exception a)  {  MessageBox.Show(a.Message, "Error encountered", MessageBoxButtons.OK);  }  zedGraphControl1.Visible = true;  }  private void metroButton2\_Click(object sender, EventArgs e)  {  clear(2);  metroButton1.Enabled = true;  metroButton2.Visible = true;  dataGridView1.Visible = true;  labelanswer.Visible = true;  zedGraphControl1.Visible = true;  metroTextBox1.Clear();  metroTextBox2.Clear();  }  private void get\_num(int x)  {  switch (x)  {  case 2:  for (int i = -30; ; i++)  {  myparse.Values["x"].SetValue(i);  xo = i;  fxo = Math.Round(myparse.Parse(metroTextBox1.Text), 4);  myparse.Values["x"].SetValue((i + 1));  x1 = i + 1;  fx1 = Math.Round(myparse.Parse(metroTextBox1.Text), 4);  if (fx1 != fxo)  {  break;  }  }  break;  }  }  public void clear(int v)  {  labelx0.Text = "Xo :";  labelx1.Text = "X1 :";  labelfx0.Text = "f(Xo) :";  labelfx1.Text = "f(X1) :";  labelanswer.Text = "Answer :";  prevnum = 0.00; test = 0.00; x2 = 0.00; x1 = 0.00; xo = 0.00; fxo = 0.0000; fx1 = 0.00;  ans = 0.00; test = 0.00;  zedGraphControl1.GraphPane.CurveList.Clear();  dataGridView1.Rows.Clear();  dataGridView1.Columns.Clear();  table();  }  public void table()  {  dataGridView1.Columns.Add("K", "K");  dataGridView1.Columns.Add("xo", "Xo");  dataGridView1.Columns.Add("x2", "X2");  dataGridView1.Columns.Add("x1", "X1");  dataGridView1.Columns.Add("fxo", "fXo");  dataGridView1.Columns.Add("fx2", "fX2");  dataGridView1.Columns.Add("fx1", "fX1");  dataGridView1.Columns.Add("error", "Error");  }  private void plot\_graph(ZedGraphControl zgc, int a)  {  // get a reference to the GraphPane  GraphPane myPane = zgc.GraphPane;  // Set the Titles  myPane.Title.Text = "Graph of Curve";  myPane.XAxis.Title.Text = "X Coordinates";  myPane.YAxis.Title.Text = "Y Coordinates";  PointPairList list1 = new PointPairList();  double y = 0;  double x;  for (int i = -25; i < 26; i++)  {  x = (double)i \* 0.5;  myparse.Values["x"].SetValue(x);  y = myparse.Parse(metroTextBox1.Text);  list1.Add(x, y);  }  LineItem myCurve = myPane.AddCurve(metroTextBox1.Text, list1, Color.Violet, SymbolType.Square);  zgc.AxisChange();  zgc.Invalidate();  zgc.Refresh();  }  }  } |

Machine Problem 3:

File: Form1.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Windows.Forms;  using System.Collections;  using MetroFramework.Forms;  using info.lundin.math;  using ZedGraph;  namespace MP3  {  public partial class Form1 : MetroForm  {  public Form1()  {  InitializeComponent();  myparse = new ExpressionParser();  myhash = new Hashtable();  table();  }  private double xo, x1, x2 = 0.00000, errornum, fxo, fx1, fx2 = 0.000, prevnum = 0.0000, ans, test;  private double x3 = 0.000, fx3 = 0.000, a, b, c, D, ho, h1, yo, y1;  private double l11, l21, l22, l31, l32, l33, u12, u13, u23, r1, r2, r3, ans1, ans2, ans3;  private void buttoncholesky2\_Click\_1(object sender, EventArgs e)  {  clear(4);  buttoncholesky2.Visible = true;  buttoncholesky1.Enabled = true;  txta11.Clear();  txta12.Clear();  txta13.Clear();  txta21.Clear();  txta22.Clear();  txta23.Clear();  txta31.Clear();  txta32.Clear();  txta33.Clear();  txtc11.Clear();  txtc21.Clear();  txtc31.Clear();  }  private void buttongauss2\_Click\_1(object sender, EventArgs e)  {  clear(5);  buttongauss2.Visible = true;  txt4a11.Clear();  txt4a12.Clear();  txt4a13.Clear();  txt4a21.Clear();  txt4a22.Clear();  txt4a23.Clear();  txt4a31.Clear();  txt4a32.Clear();  txt4a33.Clear();  txt4c11.Clear();  txt4c21.Clear();  txt4c31.Clear();  txterror5.Clear();  dataTable5.Rows.Clear();  dataTable5.Columns.Clear();  dataTable5.Visible = true;  buttongauss1.Enabled = true;  iterr1.Text = "";  iterr2.Text = "";  iterr3.Text = "";  }  private void buttongauss1\_Click(object sender, EventArgs e)  {  a11 = double.Parse(txt4a11.Text);  a12 = double.Parse(txt4a12.Text);  a13 = double.Parse(txt4a13.Text);  c11 = double.Parse(txt4c11.Text);  a21 = double.Parse(txt4a21.Text);  a22 = double.Parse(txt4a22.Text);  a23 = double.Parse(txt4a23.Text);  c21 = double.Parse(txt4c21.Text);  a31 = double.Parse(txt4a31.Text);  a32 = double.Parse(txt4a32.Text);  a33 = double.Parse(txt4a33.Text);  c31 = double.Parse(txt4c31.Text);  if (a11 > a21 && a11 > a31)  {  if (a22 > a12 && a22 > a32)  {  if (a33 > a13 && a33 > a23)  {  buttongauss2.Visible = true;  dataTable5.Visible = true;  buttongauss1.Enabled = false;  iterr1.Text = "(" + c11 + "+" + (-1) \* a12 + "y" + "+" + (-1) \* a13 + "z" + ") / (" + a11 + ")";  iterr2.Text = "(" + c21 + "+" + (-1) \* a21 + "x" + "+" + (-1) \* a23 + "z" + ") / (" + a22 + ")";  iterr3.Text = "(" + c31 + "+" + (-1) \* a31 + "x" + "+" + (-1) \* a32 + "y" + ") / (" + a33 + ")";  clear(5);  int i;  try  {  errornum = double.Parse(txterror5.Text);  for (i = 0; ; i++)  {  tempvs1 = vs1;  tempvs2 = vs2;  tempvs3 = vs3;  myhash.Clear();  myhash.Add("x", tempvs1.ToString());  myhash.Add("y", tempvs2.ToString());  myhash.Add("z", tempvs3.ToString());  vs1 = Math.Round(myparse.Parse(iterr1.Text, myhash), 4);  vs2 = Math.Round(myparse.Parse(iterr2.Text, myhash), 4);  vs3 = Math.Round(myparse.Parse(iterr3.Text, myhash), 4);  etests1 = Math.Abs(vs1 - tempvs1);  etests2 = Math.Abs(vs2 - tempvs2);  etests3 = Math.Abs(vs3 - tempvs3);  dataTable5.Rows.Add();  dataTable5.Rows[i].Cells[0].Value = i + 1;  dataTable5.Rows[i].Cells[1].Value = vs1;  dataTable5.Rows[i].Cells[2].Value = vs2;  dataTable5.Rows[i].Cells[3].Value = vs3;  dataTable5.Rows[i].Cells[4].Value = etests1;  dataTable5.Rows[i].Cells[5].Value = etests2;  dataTable5.Rows[i].Cells[6].Value = etests3;  if ((etests1 < errornum) && (etests2 < errornum) && (etests3 < errornum))  {  break;  }  }  label5X.Text = label5X.Text + Math.Round(vs1, 4).ToString();  label5Y.Text = label5Y.Text + Math.Round(vs2, 4).ToString();  label5Z.Text = label5Z.Text + Math.Round(vs3, 4).ToString();  }  catch (Exception a)  {  MessageBox.Show(a.Message, "Error encountered", MessageBoxButtons.OK);  }  }  else  {  MessageBox.Show("The dominant coefficient should be in main diagonal", "Error encountered", MessageBoxButtons.OK);  }  }  else  {  MessageBox.Show("The dominant coefficient should be in main diagonal", "Error encountered", MessageBoxButtons.OK);  }  }  else  {  MessageBox.Show("The dominant coefficient should be in main diagonal", "Error encountered", MessageBoxButtons.OK);  }  }  private void buttoncholesky1\_Click\_1(object sender, EventArgs e)  {  buttoncholesky2.Visible = true;  buttoncholesky1.Enabled = false;  try  {  clear(4);  l11 = double.Parse(txta11.Text);  l21 = double.Parse(txta21.Text);  l31 = double.Parse(txta31.Text);  u13 = double.Parse(txta12.Text) / l11;  u12 = double.Parse(txta13.Text) / l11;  l22 = double.Parse(txta22.Text) - (l21 \* u13);  l32 = double.Parse(txta32.Text) - (l31 \* u13);  u23 = (double.Parse(txta23.Text) - (l21 \* u12)) / l22;  l33 = double.Parse(txta33.Text) - (l31 \* u12) - (l32 \* u23);  r1 = double.Parse(txtc11.Text) / l11;  r2 = (double.Parse(txtc21.Text) - (l21 \* r1)) / l22;  r3 = (double.Parse(txtc31.Text) - (l32 \* r2) - (l31 \* r1)) / l33;  ans3 = r3;  ans2 = r2 - (u23 \* ans3);  ans1 = r1 - (u12 \* ans3) - (u13 \* ans2);  clabelx1.Text = clabelx1.Text + Math.Round(ans1, 4).ToString();  clabelx2.Text = clabelx2.Text + Math.Round(ans2, 4).ToString();  clabelx3.Text = clabelx3.Text + Math.Round(ans3, 4).ToString();  labL11.Text = Math.Round(l11, 2).ToString();  labL21.Text = Math.Round(l21, 2).ToString();  labL31.Text = Math.Round(l31, 2).ToString();  labL32.Text = Math.Round(l32, 2).ToString();  labL33.Text = Math.Round(l33, 2).ToString();  labL22.Text = Math.Round(l22, 2).ToString();  labU12.Text = Math.Round(u12, 2).ToString();  labU13.Text = Math.Round(u13, 2).ToString();  labU23.Text = Math.Round(u23, 2).ToString();  labC11.Text = Math.Round(double.Parse(txtc11.Text), 1).ToString();  labC21.Text = Math.Round(double.Parse(txtc21.Text), 1).ToString();  labC31.Text = Math.Round(double.Parse(txtc31.Text), 1).ToString();  labR1.Text = Math.Round(r1, 2).ToString();  labR2.Text = Math.Round(r2, 2).ToString();  labR3.Text = Math.Round(r3, 2).ToString();  }  catch (Exception a)  {  MessageBox.Show(a.Message, "Error encountered", MessageBoxButtons.OK);  }  }  private void buttonmuller2\_Click\_1(object sender, EventArgs e)  {  clear(3);  buttonmuller1.Enabled = true;  buttonmuller2.Visible = true;  dataGridView1.Visible = true;  labelanswermuller.Visible = false;  txtmuller1.Clear();  txtmuller2.Clear();  }  private void buttonmuller1\_Click(object sender, EventArgs e)  {  buttonmuller2.Visible = true;  dataGridView1.Visible = true;  labelanswermuller.Visible = true;  buttonmuller1.Enabled = false;  try  {  int x = 3, i = 0;  double tempd;  clear(3);  get\_num(x);  errornum = double.Parse(txtmuller2.Text);  labelx0.Text = labelx0.Text + xo.ToString();  labelx1.Text = labelx1.Text + x1.ToString();  labelx2.Text = labelx2.Text + x2.ToString();  labelfx0.Text = labelfx0.Text + fxo.ToString();  labelfx1.Text = labelfx1.Text + fx1.ToString();  labelfx2.Text = labelfx2.Text + fx2.ToString();  do  {  ho = Math.Round(x1 - xo, 4);  h1 = Math.Round(x2 - x1, 4);  yo = Math.Round((fx1 - fxo) / ho, 4);  y1 = Math.Round((fx2 - fx1) / h1, 4);  a = Math.Round((y1 - yo) / (h1 + ho), 4);  b = Math.Round((a \* h1) + y1, 4);  c = Math.Round(fx2, 4);  tempd = Math.Round(Math.Pow(b, 2) - (4 \* a \* c), 4);  if (tempd > 0)  {  D = Math.Round(Math.Sqrt(tempd), 4);  if (Math.Abs(b + D) > Math.Abs(b - D))  {  x3 = Math.Round(x2 - ((2 \* c) / (b + D)), 4);  }  else  {  x3 = Math.Round(x2 - ((2 \* c) / (b - D)), 4);  }  myhash.Clear();  myhash.Add("x", x3.ToString());  fx3 = Math.Round(myparse.Parse(txtmuller1.Text, myhash), 4);  test = Math.Round(Math.Abs((x3 - x2) / x3) \* 100, 4);  dataGridView1.Rows.Add();  dataGridView1.Rows[i].Cells[0].Value = i + 1;  dataGridView1.Rows[i].Cells[1].Value = xo;  dataGridView1.Rows[i].Cells[2].Value = x1;  dataGridView1.Rows[i].Cells[3].Value = x2;  dataGridView1.Rows[i].Cells[4].Value = fxo;  dataGridView1.Rows[i].Cells[5].Value = fx1;  dataGridView1.Rows[i].Cells[6].Value = fx2;  dataGridView1.Rows[i].Cells[7].Value = ho;  dataGridView1.Rows[i].Cells[8].Value = h1;  dataGridView1.Rows[i].Cells[9].Value = yo;  dataGridView1.Rows[i].Cells[10].Value = y1;  dataGridView1.Rows[i].Cells[11].Value = a;  dataGridView1.Rows[i].Cells[12].Value = b;  dataGridView1.Rows[i].Cells[13].Value = c;  dataGridView1.Rows[i].Cells[14].Value = D;  dataGridView1.Rows[i].Cells[15].Value = x3;  dataGridView1.Rows[i].Cells[16].Value = fx3;  dataGridView1.Rows[i].Cells[17].Value = test;  xo = x1; x1 = x2; x2 = x3;  fxo = fx1; fx1 = fx2; fx2 = fx3;  i++;  }  else  {  clear(3);  get\_num(x);  labelx0.Text = labelx0.Text + xo.ToString();  labelx1.Text = labelx1.Text + x1.ToString();  labelx2.Text = labelx2.Text + x2.ToString();  labelfx0.Text = labelfx0.Text + fxo.ToString();  labelfx1.Text = labelfx1.Text + fx1.ToString();  labelfx2.Text = labelfx2.Text + fx2.ToString();  test = 1000;  }  } while (test >= errornum);  ans = Math.Round(x3, 4);  labelanswermuller.Text = labelanswermuller.Text + ans.ToString();  p = 0;  }  catch (Exception a)  {  MessageBox.Show(a.Message, "Error encountered", MessageBoxButtons.OK);  }  }  private void label40\_Click(object sender, EventArgs e)  {  }  private void labL11\_Click(object sender, EventArgs e)  {  }  double a11, a12, a13, a21, a22, a23, a31, a32, a33, c11, c21, c31;  double vs1 = 0, vs2 = 0, vs3 = 0, tempvs1 = 0, tempvs2 = 0, tempvs3 = 0, etests1, etests2, etests3;  double[] col1 = new double[10];  double[] col2 = new double[10];  double[] col3 = new double[10];  private int p = 0;  private ExpressionParser myparse;  private Hashtable myhash;  private void get\_num(int x)  {  switch (x)  {  case 3:  myhash.Clear();  myhash.Add("x", (p).ToString());  xo = p;  fxo = Math.Round(myparse.Parse(txtmuller1.Text, myhash), 4);  myhash.Clear();  myhash.Add("x", (p + 1).ToString());  x1 = p + 1;  fx1 = Math.Round(myparse.Parse(txtmuller1.Text, myhash), 4);  myhash.Clear();  myhash.Add("x", (p + 2).ToString());  x2 = p + 2;  fx2 = Math.Round(myparse.Parse(txtmuller1.Text, myhash), 4);  p = p + 3;  break;  }  }  public void clear(int v)  {  if (v == 3)  {  labelx0.Text = "X0 :";  labelx1.Text = "X1 :";  labelx2.Text = "X2 :";  labelfx0.Text = "f(X0) :";  labelfx1.Text = "f(X1) :";  labelfx2.Text = "f(X2) :";  labelanswermuller.Text = "Answer :";  }  else if (v == 4)  {  clabelx1.Text = "X1 :";  clabelx2.Text = "X2 :";  clabelx3.Text = "X3 :";  labL11.Text = "L11";  labL21.Text = "L21";  labL31.Text = "L31";  labL32.Text = "L32";  labL33.Text = "L33";  labL22.Text = "L22";  labU12.Text = "U12";  labU13.Text = "U13";  labU23.Text = "U21";  labC11.Text = "C11";  labC21.Text = "C21";  labC31.Text = "C21";  labR1.Text = "R1";  labR2.Text = "R2";  labR3.Text = "R3";  }  else if (v == 5)  {  label5X.Text = "X :";  label5Y.Text = "Y :";  label5Z.Text = "Z :";  }  prevnum = 0.00; test = 0.00; x2 = 0.00; x1 = 0.00; xo = 0.00; fxo = 0.0000; fx1 = 0.00;  ans = 0.00; test = 0.00; x3 = 0.00; fx3 = 0.00;  dataGridView1.Rows.Clear();  dataGridView1.Columns.Clear();  dataTable5.Rows.Clear();  dataTable5.Columns.Clear();  table();  }  public void table()  {  dataGridView1.Columns.Add("K", "K");  dataGridView1.Columns.Add("x0", "X0");  dataGridView1.Columns.Add("x2", "X2");  dataGridView1.Columns.Add("x1", "X1");  dataGridView1.Columns.Add("fxo", "fXo");  dataGridView1.Columns.Add("fx2", "fX2");  dataGridView1.Columns.Add("fx1", "fX1");  dataGridView1.Columns.Add("ho", "ho");  dataGridView1.Columns.Add("h1", "h1");  dataGridView1.Columns.Add("yo", "yo");  dataGridView1.Columns.Add("y1", "y1");  dataGridView1.Columns.Add("a", "a");  dataGridView1.Columns.Add("b", "b");  dataGridView1.Columns.Add("c", "c");  dataGridView1.Columns.Add("D", "D");  dataGridView1.Columns.Add("x3", "X3");  dataGridView1.Columns.Add("fx3", "fX3");  dataGridView1.Columns.Add("error", "Error");  dataTable5.Columns.Add("K", "K");  dataTable5.Columns.Add("X", "X");  dataTable5.Columns.Add("Y", "Y");  dataTable5.Columns.Add("Z", "Z");  dataTable5.Columns.Add("error", "error(X)");  dataTable5.Columns.Add("error", "error(Y)");  dataTable5.Columns.Add("error", "error(Z)");  }  private void plot\_graph(ZedGraphControl zgc, int a)  {  // get a reference to the GraphPane  GraphPane myPane = zgc.GraphPane;  // Set the Titles  myPane.Title.Text = "Graph of Curve";  myPane.XAxis.Title.Text = "X Coordinates";  myPane.YAxis.Title.Text = "Y Coordinates";  PointPairList list1 = new PointPairList();  double y = 0;  double x;  for (int i = -25; i < 26; i++)  {  myhash.Clear();  x = (double)i \* 0.5;  myhash.Add("x", x.ToString());  if (a == 3)  {  y = myparse.Parse(txtmuller1.Text, myhash);  }  list1.Add(x, y);  }  LineItem myCurve = myPane.AddCurve(txtmuller1.Text, list1, Color.Violet, SymbolType.Square);  zgc.AxisChange();  zgc.Invalidate();  zgc.Refresh();  }  private void tabPage2\_Click(object sender, EventArgs e)  {  }  }  } |

Machine Problem 4:

File: Form1.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  using info.lundin.math;  using MetroFramework.Forms;  namespace MP4  {  public partial class Form1 : MetroForm  {  public Form1()  {  InitializeComponent();  }  int gridSize = 0;  int gridSizeL = 0;  int grid2Size = 0;  int gridNewtonSize = 0, gridNewton2Size = 0, gridNewton3Size = 0, gridNewton4Size = 0, gridNewton5Size = 0, gridNewton6Size = 0;  private void btnAdd\_Click(object sender, EventArgs e)  {  button2.Visible = true;  double x, y, xi2, xy;  x = double.Parse(txtX.Text);  y = double.Parse(txtFx.Text);  xi2 = x \* x;  xy = x \* y;  grid.Rows.Add(gridSize, x.ToString(), y.ToString(), xi2.ToString(), xy.ToString());  gridSize++;  txtX.Text = "";  txtFx.Text = "";  this.ActiveControl = txtX;  }  private void metroLabel23\_Click(object sender, EventArgs e)  {  }  private void metroLabel29\_Click(object sender, EventArgs e)  {  }  private void buttonclr\_Click(object sender, EventArgs e)  {  txtx1.Text = "";  txtfx1.Text = "";  txtx2.Text = "";  txtfx2.Text = "";  txtx3.Text = "";  txtfx3.Text = "";  txtx4.Text = "";  txtfx4.Text = "";  txtx5.Text = "";  txtfx5.Text = "";  listBox1.Items.Clear();  }  private void button1\_Click(object sender, EventArgs e)  {  double x1, x2, x3, x4, x5, fx1, fx2, fx3, fx4, fx5 = 0;  try  {  x1 = double.Parse(txtx1.Text);  x2 = double.Parse(txtx2.Text);  x3 = double.Parse(txtx3.Text);  x4 = double.Parse(txtx4.Text);  x5 = double.Parse(txtx5.Text);  fx1 = double.Parse(txtfx1.Text);  fx2 = double.Parse(txtfx2.Text);  fx3 = double.Parse(txtfx3.Text);  fx4 = double.Parse(txtfx4.Text);  fx5 = double.Parse(txtfx5.Text);  double f1x1, f1x2, f1x3, f1x4, f2x1, f2x2, f2x3, f3x1, f3x2, f4x1, b0, b1, b2, b3, b4 = 0;  f1x1 = Math.Round((fx2 - fx1) / (x2 - x1), 4);  f1x2 = Math.Round((fx3 - fx2) / (x3 - x2), 4);  f1x3 = Math.Round((fx4 - fx3) / (x4 - x3), 4);  f1x4 = Math.Round((fx5 - fx4) / (x5 - x4), 4);  f2x1 = Math.Round((f1x2 - f1x1) / (x3 - x1), 4);  f2x2 = Math.Round((f1x3 - f1x2) / (x4 - x2), 4);  f2x3 = Math.Round((f1x4 - f1x3) / (x5 - x3), 4);  f3x1 = Math.Round((f2x2 - f2x1) / (x4 - x1), 4);  f3x2 = Math.Round((f2x3 - f2x2) / (x5 - x1), 4);  f4x1 = Math.Round((f3x2 - f3x1) / (x5 - x1), 4);  b0 = fx1; b1 = f1x1; b2 = f2x1; b3 = f3x1;  b4 = f4x1;  double x = double.Parse(txtenterx.Text);  double answer;  listBox4.Items.Add(x1 + "\t" + fx1 + "\t" + f1x1 + "\t" + f2x1 + "\t" + f3x1 + "\t" + f4x1);  listBox4.Items.Add(x2 + "\t" + fx2 + "\t" + f1x2 + "\t" + f2x2 + "\t" + f3x2);  listBox4.Items.Add(x3 + "\t" + fx3 + "\t" + f1x3 + "\t" + f2x3);  listBox4.Items.Add(x4 + "\t" + fx4 + "\t" + f1x4);  listBox4.Items.Add(x5 + "\t" + fx5);  answer = Math.Round((b0 + (b1 \* (x - x1)) + (b2 \* (x - x1) \* (x - x2)) + (b3 \* (x - x1) \* (x - x2) \* (x - x3)) + (b4 \* (x - x1) \* (x - x2) \* (x - x3) \* (x - x4))), 4);  listBox4.Items.Add("");  listBox4.Items.Add("==============================================================================");  listBox4.Items.Add("f(" + x.ToString() + ") =" + answer.ToString());  }  catch (SystemException exa) { MessageBox.Show(exa.Message); }  }  private void button6\_Click(object sender, EventArgs e)  {  txtX.Text = "";  txtFx.Text = "";  txta0.Text = "";  txta1.Text = "";  gridSize = 0;  grid.Rows.Clear();  grid2Size = 0;  grid2.Rows.Clear();  button2.Visible = false;  }  private void button2\_Click(object sender, EventArgs e)  {  double ymm, eii, yyy, ymr, eir, yyr, sumeir, sumyyr;  LinearReg lr = new LinearReg();  lr.SetGrid(grid);  lr.solve();  lr.solveSUM();  txta0.Text = lr.getA0().ToString();  txta1.Text = lr.getA1().ToString();  int num = int.Parse(lr.getN().ToString());  double ybar = double.Parse(lr.getY().ToString());  sumeir = 0;  sumyyr = 0;  for (int i = 0; i < num; i++)  {  ymm = double.Parse(txta0.Text) + (double.Parse(txta1.Text)) \* (Double.Parse(grid[1, i].Value.ToString()));  eii = (Double.Parse(grid[2, i].Value.ToString()) - ymm) \* (Double.Parse(grid[2, i].Value.ToString()) - ymm);  yyy = (Double.Parse(grid[2, i].Value.ToString()) - ybar) \* (Double.Parse(grid[2, i].Value.ToString()) - ybar);  ymr = Math.Round(ymm, 4);  eir = Math.Round(eii, 4);  yyr = Math.Round(yyy, 4);  sumeir += eir;  sumyyr += yyr;  grid2.Rows.Add(ymr.ToString(), eir.ToString(), yyr.ToString());  }  sumX.Text = "ΣX = " + lr.getSumX().ToString();  sumY.Text = "ΣY = " + lr.getSumYY().ToString();  SumX2.Text = "ΣX^2 = " + lr.getSumX2().ToString();  SumXY.Text = "ΣXY = " + lr.getSumXXY().ToString();  SumSr.Text = "ΣSr = " + sumeir.ToString();  SumSr2.Text = "ΣSUM = " + sumyyr.ToString();  eqlbl.Text = lr.getA0().ToString() + " + " + lr.getA1().ToString() + "x";  }  private void cmdIntegrate\_Click(object sender, EventArgs e)  {  NumIntegration myClass;  myClass = new NumIntegration();  double sum, In = 0, h = 0;  double[] arx, arfx, artr;  try  {  listBox1.Items.Clear();  listBox2.Items.Clear();  listBox3.Items.Clear();  myClass.setData(int.Parse(tbox\_n.Text), double.Parse(tbox\_LLimit.Text), double.Parse(tbox\_ULimit.Text), tbox\_equation.Text.ToString());  myClass.useTrapRule(int.Parse(tbox\_n.Text), double.Parse(tbox\_LLimit.Text), double.Parse(tbox\_ULimit.Text), tbox\_equation.Text.ToString());  for (int i = 0; i < (int.Parse(tbox\_n.Text) + 1); i++)  {  arfx = myClass.getArrFX();  arx = myClass.getArrX();  artr = myClass.getArrTrap();  listBox1.Items.Add(Math.Round(arx[i], 4));  listBox2.Items.Add(Math.Round(arfx[i], 4));  listBox3.Items.Add(Math.Round(artr[i], 4));  }  sum = myClass.getSUM();  h = Math.Round((double.Parse(tbox\_ULimit.Text) - double.Parse(tbox\_LLimit.Text)) / (int.Parse(tbox\_n.Text)), 4);  In = myClass.getI();  hlbl.Text = h.ToString();  SUMlbl.Text = sum.ToString();  Ilbl.Text = In.ToString();  }  catch (FormatException a)  {  MessageBox.Show(a.Message, "Error Display", MessageBoxButtons.OK, MessageBoxIcon.Error);  }  }  private void btnClear\_Click(object sender, EventArgs e)  {  listBox1.Items.Clear();  listBox2.Items.Clear();  listBox3.Items.Clear();  tbox\_equation.Clear();  tbox\_LLimit.Clear();  tbox\_n.Clear();  tbox\_ULimit.Clear();  }  private void button5\_Click(object sender, EventArgs e)  {  NumDifferentiation myDiff = new NumDifferentiation();  try  {  myDiff.Start(double.Parse(tbox\_h.Text), double.Parse(tbox\_Xi.Text));  myDiff.Compute(tbox\_equations.Text);  FFDD.Visible = true;  myDiff.ForwardDivideDifference(tbox\_equations.Text.ToString());  tbox\_diff.Text = myDiff.Display();  tbox\_diffMore.Text = Math.Round(myDiff.getDiffEstiMORE(), 4).ToString();  XilblF.Text = tbox\_Xi.Text;  FXilblF.Text = Math.Round(myDiff.getFX(), 4).ToString();  Xi1lblF.Text = Math.Round(myDiff.getXi1(), 4).ToString();  FXi1lblF.Text = Math.Round(myDiff.getFXi1(), 4).ToString();  Xi2lblF.Text = Math.Round(myDiff.getXi2(), 4).ToString();  FXi2lblF.Text = Math.Round(myDiff.getFXi2(), 4).ToString();  }  catch (FormatException E)  {  MessageBox.Show(E.Message, "Error Display", MessageBoxButtons.OK, MessageBoxIcon.Error);  }  }  private void button11\_Click(object sender, EventArgs e)  {  tbox\_Xi.Clear();  tbox\_equations.Clear();  tbox\_diff.Clear();  tbox\_h.Clear();  }  }  }  **LinearReg.cs:**  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace MP4  {  class LinearReg  {  private System.Windows.Forms.DataGridView grid;  int n = 0;  double sumx = 0.0;  double sumy = 0.0;  double sumxy = 0.0;  double a0 = 0.0;  double a1 = 0.0;  double sumyy = 0.0;  double sume2 = 0.0;  double r = 0.0;  double r1 = 0.0;  double exp = 0.5;  double temp = 0.0;  double ybar = 0.0;  int i = 0;  double sumxx = 0, sumyyy = 0, sumx2 = 0;  double sumxxy = 0;  public void SetGrid(System.Windows.Forms.DataGridView g)  {  grid = g;  }  public void solve()  {  n = grid.Rows.Count;  sumx = 0;  for (i = 0; i < n; i++)  {  sumx += Double.Parse(grid[1, i].Value.ToString());  }  for (i = 0; i < n; i++)  {  sumy += Double.Parse(grid[2, i].Value.ToString());  }  for (i = 0; i < n; i++)  {  temp = Double.Parse(grid[1, i].Value.ToString()) \* Double.Parse(grid[2, i].Value.ToString());  sumxy += Math.Round(temp, 5);  }  for (i = 0; i < n; i++)  {  temp = Double.Parse(grid[1, i].Value.ToString()) \* Double.Parse(grid[1, i].Value.ToString());  sume2 += Math.Round(temp, 5);  }  a1 = (n \* sumxy - sumx \* sumy) / (7 \* sume2 - (sumx \* sumx));  a1 = Math.Round(a1, 5);  a0 = (sumy / n) - a1 \* (sumx / n);  a0 = Math.Round(a0, 5);  ybar = sumy / n;  for (i = 0; i < n; i++)  {  temp = (Double.Parse(grid[2, i].Value.ToString()) - ybar) \* (Double.Parse(grid[2, i].Value.ToString()) - ybar);  sumyy += Math.Round(temp, 5);  }  for (i = 0; i < n; i++)  {  temp = Double.Parse(grid[2, i].Value.ToString()) - a0 - a1 \* Double.Parse(grid[1, i].Value.ToString());  sume2 += Math.Round(temp \* temp, 5);  }  r = ((sumyy - sume2) / sumyy);  r1 = (Math.Pow(r, exp));  }  public void solveSUM()  {  n = grid.Rows.Count;  for (int i = 0; i < n; i++)  {  sumxx += Double.Parse(grid[1, i].Value.ToString());  sumyyy += Double.Parse(grid[2, i].Value.ToString());  sumx2 += Double.Parse(grid[3, i].Value.ToString());  sumxxy += Double.Parse(grid[4, i].Value.ToString());  }  }  public double getA0()  {  return a0;  }  public double getA1()  {  return a1;  }  public double getSumX()  {  return sumxx;  }  public double getSumYY()  {  return sumyyy;  }  public double getSumX2()  {  return sumx2;  }  public double getSumXXY()  {  return sumxxy;  }  public double getY()  {  return ybar;  }  public String getR()  {  return r1.ToString("F4");  }  public double getN()  {  n = grid.RowCount;  return n;  }  }  }  **NumDifferentiation.cs:**  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Collections;  using info.lundin.math;  namespace MP4  {  class NumDifferentiation  {  private double xi;  private double fxi;  private double xiadd;  private double fxiadd;  private double ximinus;  private double fximinus;  private double xiadd2;  private double fxiadd2;  private double ximinus2;  private double fximinus2;  private double h;  private double diffEstimate;  private double diffEstimateMORE;  private ExpressionParser myParse;  Hashtable myHash;  public void Start(double stepSize, double inSize)  {  myParse = new ExpressionParser();  myHash = new Hashtable();  xi = inSize;  h = stepSize;  }  public double getDiffEstiMORE()  {  return diffEstimateMORE;  }  public double getFX()  {  return fxi;  }  public double getXi1()  {  return xiadd;  }  public double getFXi1()  {  return fxiadd;  }  public double getXi2()  {  return xiadd2;  }  public double getFXi2()  {  return fxiadd2;  }  public double getFXmin1()  {  return fximinus;  }  public double getXmin1()  {  return ximinus;  }  public double getFXmin2()  {  return fximinus2;  }  public double getXmin2()  {  return ximinus2;  }  public void Compute(string equation)  {  xiadd = xi + h;  xiadd2 = xi + (2 \* h);  ximinus = xi - h;  ximinus2 = xi - (2 \* h);  myHash.Clear();  myHash.Add("x", xi.ToString());  fxi = myParse.Parse(equation, myHash);  myHash.Clear();  myHash.Add("x", xiadd.ToString());  fxiadd = myParse.Parse(equation, myHash);  myHash.Clear();  myHash.Add("x", xiadd2.ToString());  fxiadd2 = myParse.Parse(equation, myHash);  myHash.Clear();  myHash.Add("x", ximinus.ToString());  fximinus = myParse.Parse(equation, myHash);  myHash.Clear();  myHash.Add("x", ximinus2.ToString());  fximinus2 = myParse.Parse(equation, myHash);  }  public void ForwardDivideDifference(string equation)  {  diffEstimate = (fxiadd - fxi) / (xiadd - xi);  diffEstimateMORE = ((-1 \* fxiadd2) + (4 \* fxiadd) + (-3 \* fxi)) / (2 \* h);  }  public string Display()  {  //string message = "Estimate: " + diffEstimate;  string message = Math.Round(diffEstimate, 5).ToString();  return message;  }  }  }  **NumIntegration.cs:**  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Collections;  using info.lundin.math;  namespace MP4  {  class NumIntegration  {  double h, f, r, sum = 0, I = 0, IR = 0;  private ExpressionParser myParse;  private Hashtable myHash;  double[] arrx = new double[10];  double[] arrfx = new double[10];  double[] arrtrap = new double[10];  public double[] getArrFX()  {  return arrfx;  }  public double[] getArrX()  {  return arrx;  }  public double[] getArrTrap()  {  return arrtrap;  }  public double getSUM()  {  return sum;  }  public double getI()  {  return IR;  }  public void setData(int n, double xa, double xb, string equation)  {  myParse = new ExpressionParser();  myHash = new Hashtable();  h = (xb - xa) / n;  for (int i = 0; i < (n + 1); i++)  {  if (i == 0)  f = (xa);  else  f = f + h;  arrx[i] = Math.Round(f, 4);  }  for (int j = 0; j < (n + 1); j++)  {  if (j == 0)  f = (xa);  else  myHash.Clear();  myHash.Add("x", arrx[j].ToString());  arrfx[j] = Math.Round(myParse.Parse(equation, myHash), 4);  }  }  public void useTrapRule(int n, double xa, double xb, string equation)  {  myParse = new ExpressionParser();  myHash = new Hashtable();  h = (xb - xa) / n;  for (int i = 0; i < (n + 1); i++)  {  if (i == 0)  f = (xa);  else  f = f + h;  arrx[i] = Math.Round(f, 4);  }  for (int j = 0; j < (n + 1); j++)  {  if (j == 0)  f = (xa);  else  myHash.Clear();  myHash.Add("x", arrx[j].ToString());  arrfx[j] = Math.Round(myParse.Parse(equation, myHash), 4);  }  for (int k = 0; k < (n + 1); k++)  {  if (k == 0)  f = (xa);  else  getArrFX();  if (k == 0 || k == n)  r = (1 \* arrfx[k]);  else  {  r = (2 \* arrfx[k]);  }  arrtrap[k] = Math.Round(r, 4);  sum += arrtrap[k];  }  // getSUM();  I = (h / 2) \* sum;  IR = Math.Round(I, 4);  }  }  } |

**REFERENCES\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Numerical Method for Engineers, 6th

Ed., by Steven C. Chapra and

Raymond P. Canale (Book)

<http://stattrek.com/regression/linear-regression.aspx>

<https://ariacarmen10.wordpress.com/2012/03/15/c-newtons-divided-difference/>

<http://www.che.utah.edu/~aabboud/2450/Additional/Derivatives.pdf>

http://pages.pacificcoast.net/~cazelais/187/trapezoidal\_rule.pdf

info.lundin.math.dll

MetroFramework.dll

MetroFramework.Design.dll

MetroFramework.Fonts.dll