Mapúa Institute of Technology

School of Electrical, Electronics, and Computer Engineering

Machine Problem 1

Numerical Methods

Andrada, Luke Clark M.
COE60 C1

Engr. Carlos Hortinela IV
March 30, 2016

DISCUSSION

BRACKETING TECHNIQUE
REGULA-FALSI METHOD

The *Regula-Falsi Method* or *False Position Method* is a numerical method employing bracketing technique for approximating the roots of polynomial or transcendental functions.

It begins with two (2) initial values x_0 and x_1 such that when substituted to the function deliver results $f(x_0)$ and $f(x_1)$ of contrary signs implying that the initial values bracket the root of the function within that interval. The interval shortens per iteration as the process approximates the root of the function.

A value x_2 is calculated with a formula dependent on the values x_0 and x_1 and is substituted to the value that delivers the same sign as the calculated value when substituted to the function. The aforesaid process of calculation and substitution will repeat itself until the defined error tolerance e has been satisfied.

The formula for calculating the value x_2 is

$$x_2 = x_0 - f(x_0)\left[\frac{x_1 - x_0}{f(x_1) - f(x_0)}\right]$$

or

$$x_2 = x_1 - f(x_1)\left[\frac{x_1 - x_0}{f(x_1) - f(x_0)}\right]$$

A table can be drafted to summarize the process such as

x_0	x_2	x_1	$f(x_0)$	$f(x_2)$	$f(x_1)$

The formula for calculating the error e is

$$e = |x_2' - x_2|$$

where x_2' is the latest calculated value and x_2 the older.

It can further be noted that $f(x_0)$ and $f(x_1)$ cannot be the same because the process will encounter division by zero when so.

In general, the aforesaid method executes as follows

1) Identify the initial values x_0 and x_1
2) Solve for $f(x_0)$ and $f(x_1)$
3) Solve for x_2
4) Substitute x_2 to x_0 or x_1
5) Solve for e
6) Repeat

.

USER MANUAL

BRACKETING TECHNIQUE
REGULA-FALSI METHOD

The steps for successful utilization of the program are as follows

1) Open MP1

   The program opens with an introductory window containing the button that will open the selected method.

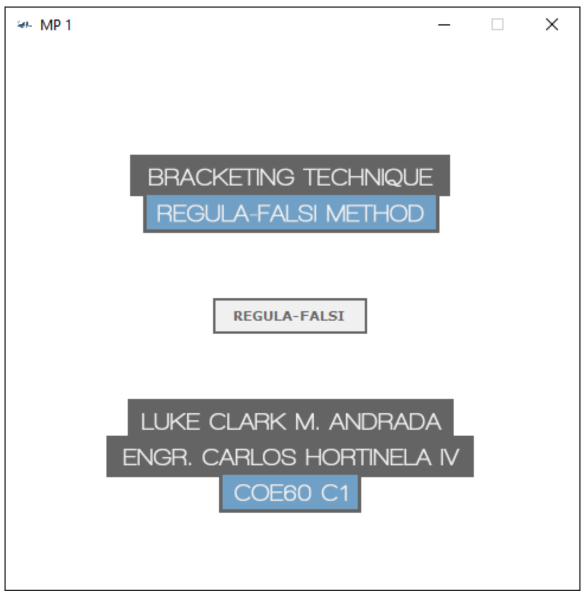

FIG 1. MENU WINDOW

2) Click REGULA-FALSI

   The button opens another window containing the input and output elements of the selected method.

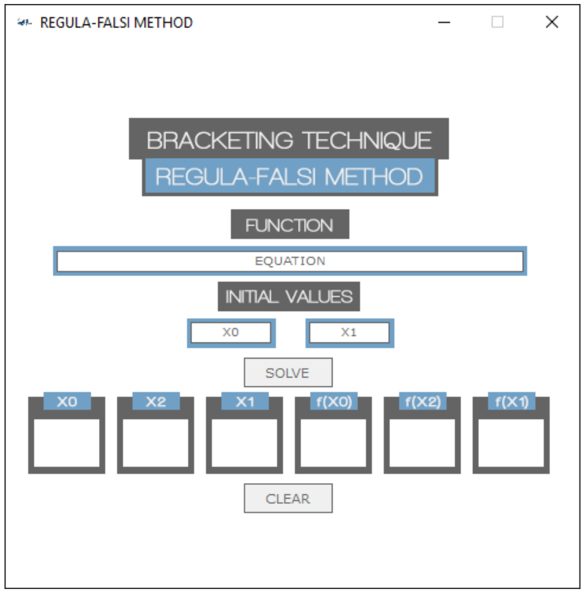

FIG 2. REGULA-FALSI WINDOW

3) Enter EQUATION

The program has the capability to accept both polynomials and transcendental functions and further supports several operators, functions, and constants with the help of an external math parser. It can accept and support the following

OPERATORS

+	-	*	/	^	%

FUNCTIONS

SQRT	SIN	COS	TAN	ATAN	ACOS	ASIN	SINH	COSH	TANH	ACOTAN
EXP	LN	LOG	ABS	CIEL	FAC	SFAC	ROUND	FLOOR	FPART	

CONSTANTS

PI	EULER	FALSE	INFINITY

The correct syntax can be consulted at lundin.info/mathparser.

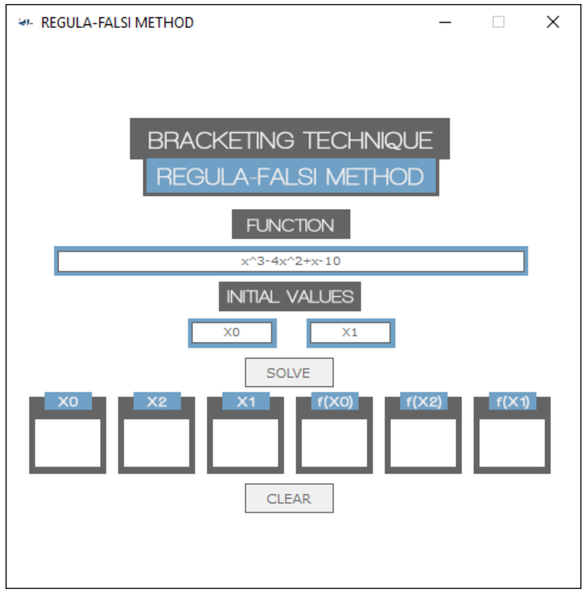

FIG 3. ENTER FUNCTION EX. $x^3 - 4x^2 + x - 10$

4) Enter INITIAL VALUES

The initial values can be identified by trial and error as per the rules of the method but the program has the capability to accept any initial values at the cost of iteration overhead.

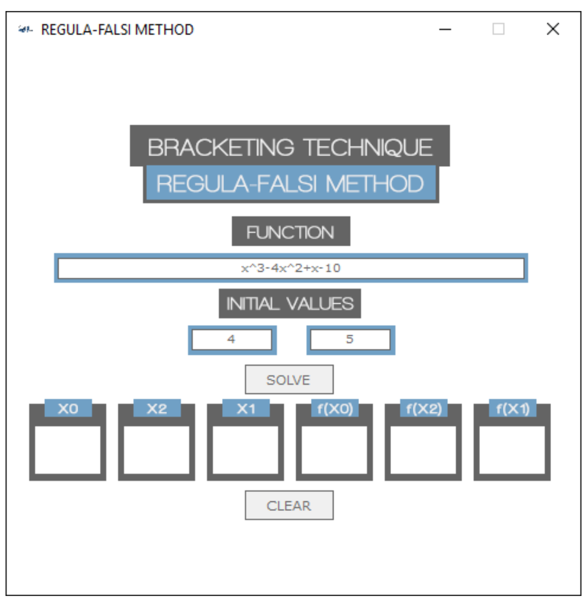

FIG 4. ENTER INITIAL VALUES EX. 4 AND 5

5) Click SOLVE

The program solves for the root of the function implementing the rules of the selected method, drafts the table, and displays the result.

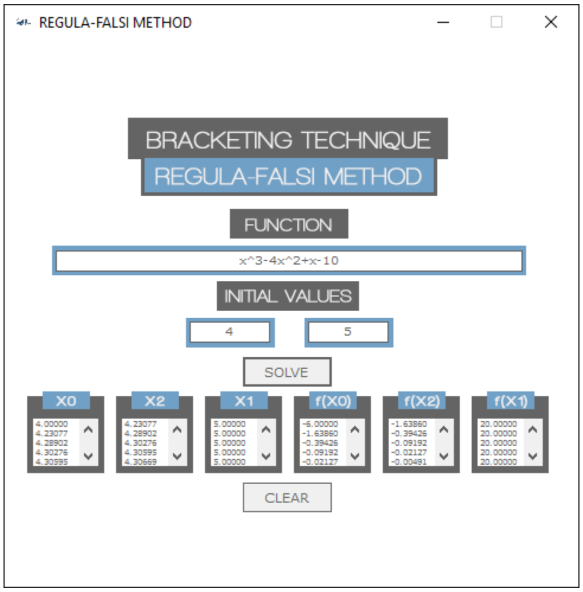

FIG 5. TABLE DRAFT

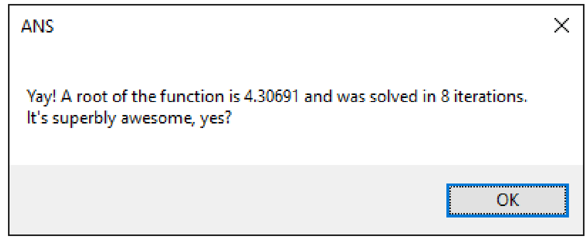

FIG 6. RESULT DISPLAY

6) Click CLEAR

The program resets the window and its elements to start over.

The other specifications of the program are as follows:

- It works in five (5) decimal places and double data type.
- It limits the iteration to one thousand (1000).
- It handles no-input, invalid-input, zero-division, nan-result, and inf-result calculation and programming errors.

APPENDIX

BRACKETING TECHNIQUE
REGULA-FALSI METHOD

SOURCE CODE

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;

/*
    LUKE CLARK M. ANDRADA

    ENGR. CARLOS HORTINELA IV
    COE60 C1

    MP1
    BRACKETING METHOD x REGULA-FALSI METHOD

    REFERENCES
    lundin.info/mathparser
    stackoverflow.com
    msdn.microsoft.com
    existing applications
*/

namespace MP1
{
    public partial class frmMain : Form
    {
        public frmMain()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void btnRegula_Click(object sender, EventArgs e)
        {
            MP1.frmRegula form = new MP1.frmRegula();
            form.ShowDialog();
        }
    }
}
```

FORM2.CS

```
using System;
using System.Collections.Generic;
using System.Collections;
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;

namespace MP1
{
   public partial class frmRegula : Form
   {
      // globally initialize variables
      int count;
      double error = 0.00001;
      double x0, x1, x2 = 9999, t;
      double fx0, fx1, fx2, temp;
      int test;

      ExpressionParser myParse;
      Hashtable myHash;

      public frmRegula()
      {
         InitializeComponent();
      }

      public int check()
      {
         // solve for fx0 and fx1
         fx0 = fx1 = fx2 = 0;

         // handle errors
         try
         {
            myHash.Clear();
            myHash.Add("x", x0.ToString());
            fx0 = myParse.Parse(txtBoxRegula_eq.Text, myHash);

            myHash.Clear();
            myHash.Add("x", x1.ToString());
            fx1 = myParse.Parse(txtBoxRegula_eq.Text, myHash);

            // check for Nan or inf
            if (double.IsNaN(fx0) || double.IsNaN(fx1) || double.IsInfinity(fx0) || double.IsInfinity(fx0))
            {
               MessageBox.Show("Hi. I'm sorry but with the given function and initial values,\nI can tell you that the result will be
NaN and I can't handle NaN.\nI want you to try again, okay?", "ERR", MessageBoxButtons.OK);
               return 1;
            }

            // check if fx0 and fx1 are the same
            if (fx0 == fx1)
```

```
            {
                MessageBox.Show("Hi. I'm sorry but the f(x)'s of the initial values can't be the same.\nIf so, we'll be diving by zero and ending the world.\nI want you try again, okay?", "ERR", MessageBoxButtons.OK);
                return 1;
            }
        }

        catch
        {
            MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR", MessageBoxButtons.OK);
            return 1;
        }

        return 0;
    }

    private void btnRegula_solve_Click(object sender, EventArgs e)
    {
        // iteration counter
        count = 0;

        x2 = 9999;

        // handle errors
        try
        {
            myParse = new ExpressionParser();
            myHash = new Hashtable();

            x0 = double.Parse(txtBoxRegula_int1.Text);
            x1 = double.Parse(txtBoxRegula_int2.Text);
        }

        catch (FormatException)
        {
            clearBox();
            MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR", MessageBoxButtons.OK);
            return;
        }

        // clear textboxes
        clearBox();

        // exit if the initial values are invalid
        test = check();

        if (test != 0)
            return;

        do
        {
            // write x0 and x1
            count++;
            fx0 = fx1 = fx2 = 0;

            lBoxRegula_x0.Items.Add(Math.Round(x0, 5).ToString("0.00000"));
```

```
            lBoxRegula_x1.Items.Add(Math.Round(x1, 5).ToString("0.00000"));

            // solve for fx0 and fx1
            myHash.Clear();
            myHash.Add("x", x0.ToString());
            fx0 = myParse.Parse(txtBoxRegula_eq.Text, myHash);

            myHash.Clear();
            myHash.Add("x", x1.ToString());
            fx1 = myParse.Parse(txtBoxRegula_eq.Text, myHash);

            // write fx0 and fx1
            lBoxRegula_fx0.Items.Add(Math.Round(fx0, 5).ToString("0.00000"));
            lBoxRegula_fx1.Items.Add(Math.Round(fx1, 5).ToString("0.00000"));

            temp = x2;

            // check for Nan or inf
            if (double.IsNaN(fx0) || double.IsNaN(fx1) || double.IsInfinity(fx0) || double.IsInfinity(fx0))
            {
              clearBox();
              MessageBox.Show("Hi. I'm sorry but with the given function and initial values,\nI can tell you that the result will be
NaN and I can't handle NaN.\nI want you to try again, okay?", "ERR", MessageBoxButtons.OK);
              return;
            }

            // check if fx0 and fx1 are the same
            if (fx0 == fx1)
            {
              clearBox();
              MessageBox.Show("Hi. I'm sorry but the f(x)'s of the x values at iteration " + count + " are the same.\nIf so, we'll be
diving by zero and ending the world.\nI want you try again, okay?", "ERR", MessageBoxButtons.OK);
              return;
            }

            // solve and write x2
            x2 = x1 - fx1 * ((x1 - x0) / (fx1 - fx0));
            lBoxRegula_x2.Items.Add(Math.Round(x2, 5).ToString("0.00000"));

            myHash.Clear();
            myHash.Add("x", x2.ToString());
            fx2 = myParse.Parse(txtBoxRegula_eq.Text, myHash);

            lBoxRegula_fx2.Items.Add(Math.Round(fx2, 5).ToString("0.00000"));

            // swap values aptly
            t = fx0 * fx2;

            if (t > 0)
              x0 = x2;
            else if (t < 0)
              x1 = x2;
          } while ((Math.Abs(temp - x2)) >= error && count < 1000);

          if(count == 1000)
          {
            clearBox();
```

```
            MessageBox.Show("Hi. I'm sorry but I can't solve it.\nI tried but the iterations were over a thousand!", "ERR",
MessageBoxButtons.OK);
            return;
        }

        // display result
        MessageBox.Show("Yay! A root of the function is " + Math.Round(x2, 5).ToString("0.00000") + " and was solved in " + count
+ " iterations.\nIt's superbly awesome, yes?", "ANS", MessageBoxButtons.OK);
    }

    private void btnRegula_clear_Click(object sender, EventArgs e)
    {
        // clear everything
        txtBoxRegula_int1.Text = "X0";
        txtBoxRegula_int2.Text = "X1";
        txtBoxRegula_eq.Text = "EQUATION";

        clearBox();
    }

    private void clearBox()
    {
        lBoxRegula_x0.Items.Clear();
        lBoxRegula_x2.Items.Clear();
        lBoxRegula_x1.Items.Clear();
        lBoxRegula_fx0.Items.Clear();
        lBoxRegula_fx2.Items.Clear();
        lBoxRegula_fx1.Items.Clear();
    }
  }
}
```

APPENDIX

REFERENCES

lundin.info/mathparser

stackoverflow.com

msdn.microsoft.com

existing applications

Mapúa Institute of Technology

School of Electrical, Electronics, and Computer Engineering

Machine Problem 2

Numerical Methods

Andrada, Luke Clark M.
COE60 C1

Engr. Carlos Hortinela IV
March 30, 2016

DISCUSSION

OPEN METHOD

SECANT METHOD

The *Secant Method* is a numerical method for better approximating the roots of a polynomial or transcendental function by employing a succession of roots of secant lines.

It begins with two (2) initial values x_0 and x_1 that can be identified at the will of the user hence the label *Open Method*. A value x_2 is calculated with a formula dependent on x_0 and x_1 and is substituted to the latter while the latter is substituted to the former and discarding its former value.

The value x_2 approaches the root of the function as the aforesaid process of calculation and substitution repeats itself and will only terminate when the defined error tolerance e has been satisfied.

The formula for calculating the value x_2 is

$$x_2 = x_0 - f(x_0)\left[\frac{x_1 - x_0}{f(x_1) - f(x_0)}\right]$$

or

$$x_2 = x_1 - f(x_1)\left[\frac{x_1 - x_0}{f(x_1) - f(x_0)}\right]$$

A table can be drafted to summarize the process such as

x_0	x_2	x_1	$f(x_0)$	$f(x_2)$	$f(x_1)$

The formula for calculating the error e is

$$e = |x_2' - x_2|$$

where x_2' is the latest calculated value and x_2 the older.

It can further be noted that $f(x_0)$ and $f(x_1)$ cannot be the same because the process will encounter division by zero when so.

In general, the aforesaid method executes as follows

1) Identify the initial values x_0 and x_1
2) Solve for $f(x_0)$ and $f(x_1)$
3) Solve for x_2
4) Substitute x_1 to x_0 and x_2 to x_1
5) Solve for e
6) Repeat

USER MANUAL

OPEN METHOD
SECANT METHOD

The steps for successful utilization of the program are as follows

1) Open MP2

    The program opens with an introductory window containing the button that will open the selected method.

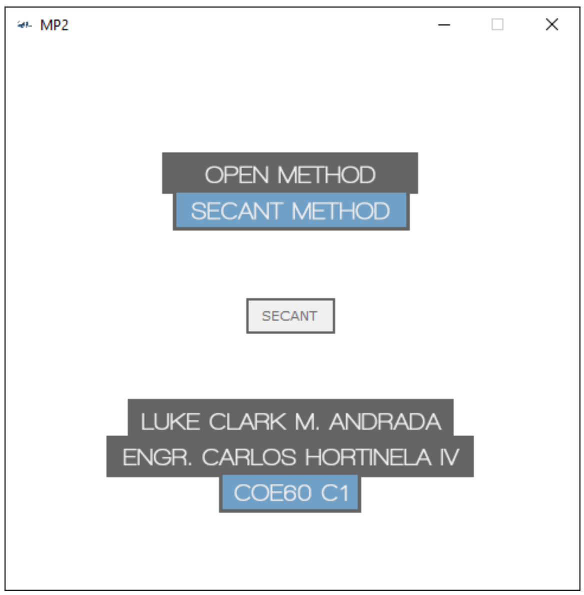

FIG 1. MENU WINDOW

2) Click SECANT

    The button opens another window containing the input and output elements of the selected method.

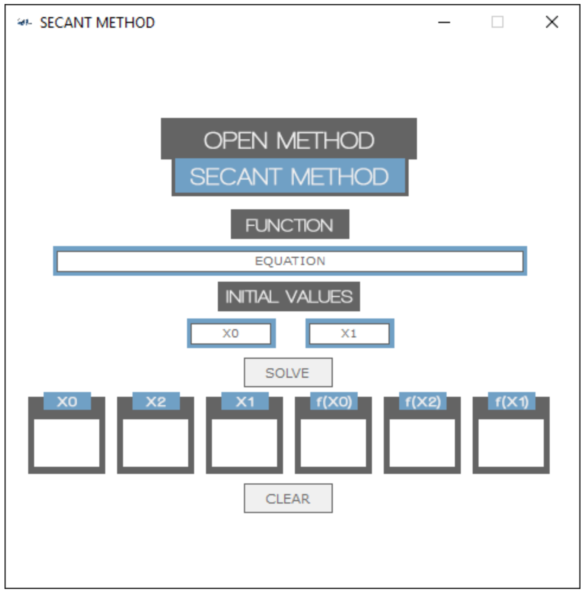

FIG 2. SECANT WINDOW

3) Enter EQUATION

The program has the capability to accept both polynomials and transcendental functions and further supports several operators, functions, and constants with the help of an external math parser. It can accept and support the following

OPERATORS

+	-	*	/	^	%

FUNCTIONS

SQRT	SIN	COS	TAN	ATAN	ACOS	ASIN	SINH	COSH	TANH	ACOTAN
EXP	LN	LOG	ABS	CIEL	FAC	SFAC	ROUND	FLOOR	FPART	

CONSTANTS

PI	EULER	FALSE	INFINITY

The correct syntax can be consulted at lundin.info/mathparser.

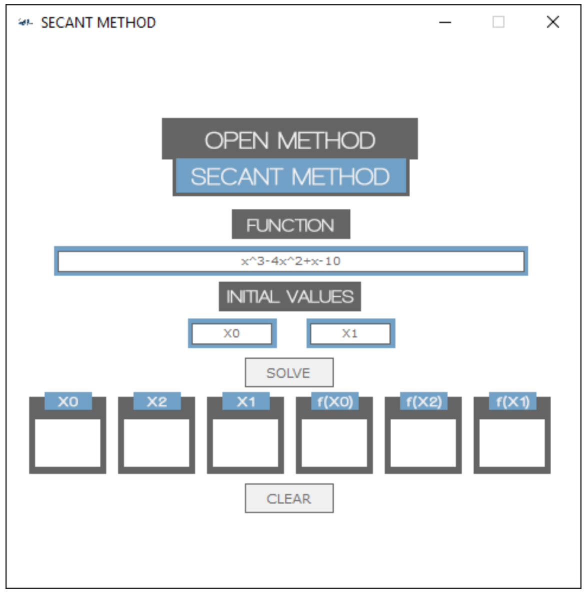

FIG 3. ENTER FUNCTION EX. $x^3 - 4x^2 + x - 10$

4) Enter INITIAL VALUES

The initial values can be identified at the will of the user.

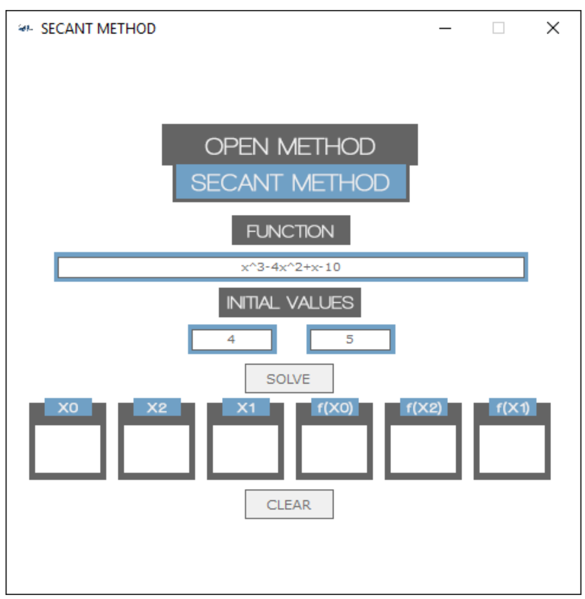

FIG 4. ENTER INITIAL VALUES EX. 4 AND 5

5) Click SOLVE

The program solves for the root of the function implementing the rules of the selected method, drafts the table, and displays the result.

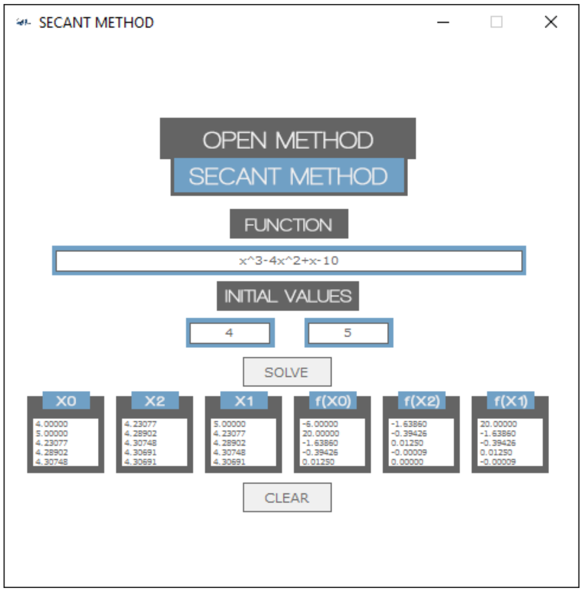

FIG 5. TABLE DRAFT

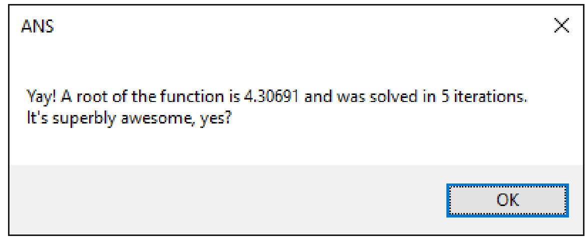

FIG 6. RESULT DISPLAY

6) Click CLEAR

The program resets the window and its elements to start over.

The other specifications of the program are as follows:

- It works in five (5) decimal places and double data type.
- It limits the iteration to one thousand (1000).
- It handles no-input, invalid-input, zero-division, nan-result, and inf-result calculation and programming errors.

APPENDIX

OPEN METHOD
SECANT METHOD

SOURCE CODE

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;

/*
   LUKE CLARK M. ANDRADA

   ENGR. CARLOS HORTINELA IV
   COE60 C1

   MP2
   OPEN METHOD x SECANT METHOD

   REFERENCES
   lundin.info/mathparser
   stackoverflow.com
   msdn.microsoft.com
   existing applications
*/

namespace MP2
{
   public partial class frmMain : Form
   {
      public frmMain()
      {
         InitializeComponent();
      }

      private void btnSecant_Click(object sender, EventArgs e)
      {
         MP2.frmSecant form = new MP2.frmSecant();
         form.ShowDialog();
      }
   }
}
```

FORM2.CS

```
using System;
using System.Collections.Generic;
using System.Collections;
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;

namespace MP2
{
  public partial class frmSecant : Form
  {
    // globally initialize variables
    int count;
    double error = 0.00001;
    double x0, x1, x2 = 9999;
    double fx0, fx1, fx2, temp;
    int test;

    ExpressionParser myParse;
    Hashtable myHash;

    public frmSecant()
    {
      InitializeComponent();
    }

    public int check()
    {
      // solve for fx0 and fx1
      fx0 = fx1 = fx2 = 0;

      // handle errors
      try
      {
        myHash.Clear();
        myHash.Add("x", x0.ToString());
        fx0 = myParse.Parse(txtBoxSecant_eq.Text, myHash);

        myHash.Clear();
        myHash.Add("x", x1.ToString());
        fx1 = myParse.Parse(txtBoxSecant_eq.Text, myHash);

        // check for Nan or inf
        if (double.IsNaN(fx0) || double.IsNaN(fx1) || double.IsInfinity(fx0) || double.IsInfinity(fx0))
        {
          MessageBox.Show("Hi. I'm sorry but with the given function and initial values,\nI can tell you that the result will be
NaN and I can't handle NaN.\nI want you to try again, okay?", "ERR", MessageBoxButtons.OK);
          return 1;
        }

        // check if fx0 and fx1 are the same
        if (fx0 == fx1)
```

```
            {
                MessageBox.Show("Hi. I'm sorry but the f(x)'s of the initial values can't be the same.\nIf so, we'll be diving by zero and ending the world.\nI want you try again, okay?", "ERR", MessageBoxButtons.OK);
                return 1;
            }
        }

        catch
        {
            MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR", MessageBoxButtons.OK);
            return 1;
        }

        return 0;
    }

    private void btnSecant_solve_Click(object sender, EventArgs e)
    {
        // iteration counter
        count = 0;

        x2 = 9999;

        // handle format errors
        try
        {
            myParse = new ExpressionParser();
            myHash = new Hashtable();

            x0 = double.Parse(txtBoxSecant_int1.Text);
            x1 = double.Parse(txtBoxSecant_int2.Text);
        }

        catch (FormatException)
        {
            clearBox();
            MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR", MessageBoxButtons.OK);
            return;
        }

        // clear textboxes
        clearBox();

        // exit if the initial values are invalid
        test = check();

        if (test != 0)
            return;

        do
        {
            // write x0 and x1
            count++;
            fx0 = fx1 = fx2 = 0;

            lBoxSecant_x0.Items.Add(Math.Round(x0, 5).ToString("0.00000"));
```

```
            lBoxSecant_x1.Items.Add(Math.Round(x1, 5).ToString("0.00000"));

            // solve for fx0 and fx1
            myHash.Clear();
            myHash.Add("x", x0.ToString());
            fx0 = myParse.Parse(txtBoxSecant_eq.Text, myHash);

            myHash.Clear();
            myHash.Add("x", x1.ToString());
            fx1 = myParse.Parse(txtBoxSecant_eq.Text, myHash);

            // write fx0 and fx1
            lBoxSecant_fx0.Items.Add(Math.Round(fx0, 5).ToString("0.00000"));
            lBoxSecant_fx1.Items.Add(Math.Round(fx1, 5).ToString("0.00000"));

            temp = x2;

            // check for Nan or inf
            if (double.IsNaN(fx0) || double.IsNaN(fx1) || double.IsInfinity(fx0) || double.IsInfinity(fx0))
            {
              clearBox();
              MessageBox.Show("Hi. I'm sorry but with the given function and initial values,\nI can tell you that the result will be
NaN and I can't handle NaN.\nI want you to try again, okay?", "ERR", MessageBoxButtons.OK);
              return;
            }

            // check if fx0 and fx1 are the same
            if (fx0 == fx1)
            {
              clearBox();
              MessageBox.Show("Hi. I'm sorry but the f(x)'s of the x values at iteration " + count + " are the same.\nIf so, we'll be
diving by zero and ending the world.\nI want you try again, okay?", "ERR", MessageBoxButtons.OK);
              return;
            }

            // solve and write x2
            x2 = x1 - fx1 * ((x1 - x0) / (fx1 - fx0));
            lBoxSecant_x2.Items.Add(Math.Round(x2, 5).ToString("0.00000"));

            // solve and write fx2
            myHash.Clear();
            myHash.Add("x", x2.ToString());
            fx2 = myParse.Parse(txtBoxSecant_eq.Text, myHash);

            lBoxSecant_fx2.Items.Add(Math.Round(fx2, 5).ToString("0.00000"));

            // swap values
            x0 = x1;
            x1 = x2;

          } while ((Math.Abs(temp - x2)) >= error && count < 1000);

          if (count == 1000)
          {
            MessageBox.Show("Hi. I'm sorry but I can't solve it.\nI tried but the iterations were over a thousand!", "ERR",
MessageBoxButtons.OK);
            return;
          }
```

```
            // display result
            MessageBox.Show("Yay! A root of the function is " + Math.Round(x2, 5).ToString("0.00000") + " and was solved in " + count
+ " iterations.\nIt's superbly awesome, yes?", "ANS", MessageBoxButtons.OK);
        }

        private void btnSecant_clear_Click(object sender, EventArgs e)
        {
            // clear everything
            txtBoxSecant_int1.Text = "X0";
            txtBoxSecant_int2.Text = "X1";
            txtBoxSecant_eq.Text = "EQUATION";

            clearBox();
        }

        private void clearBox()
        {
            lBoxSecant_x0.Items.Clear();
            lBoxSecant_x2.Items.Clear();
            lBoxSecant_x1.Items.Clear();
            lBoxSecant_fx0.Items.Clear();
            lBoxSecant_fx2.Items.Clear();
            lBoxSecant_fx1.Items.Clear();
        }
    }
}
```

APPENDIX

REFERENCES

lundin.info/mathparser

stackoverflow.com

msdn.microsoft.com

existing applications

Mapúa Institute of Technology
School of Electrical, Electronics, and Computer Engineering

Machine Problem 3
Numerical Methods

Andrada, Luke Clark M.
COE60 C1

Engr. Carlos Hortinela IV
March 30, 2016

DISCUSSION

POLYNOMIAL TECHNIQUE
MULLER'S METHOD

The *Muller's Method* is a numerical method for approximating the roots of a polynomial function by constructing a parabola and taking its intersection with the x-axis as the next approximation.

It begins with three (3) initial values x_0, x_1, and x_2 selected at the will of the user where the parabola will pass through. The curve-fitting values δ_0 and δ_1 which are slopes between the aforesaid points, and h_0 and h_1 which are distances between the aforesaid values are derived to solve for the coefficients of the quadratic model function that in turn will be used to solve for the next approximation x_3.

The value x_3 is then substituted to x_2 and x_2 to x_1 and so forth. The process will repeat itself until the defined error tolerance e has been satisfied.

The quadratic model function is

$$f_2(x) = a(x - x_2)^2 + b(x - x_2) + c$$

The formulas for calculating the value of the curve-fitting values are

$$\delta_0 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

$$\delta_1 = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

$$h_0 = x_1 - x_0$$

$$h_1 = x_2 - x_1$$

The formulas for calculating the value of the coefficients are

$$a = \frac{\delta_1 - \delta_0}{h_1 - h_0}$$

$$b = ah_1 + \delta_1$$

$$c = f(x_2)$$

The formula for calculating the value x_3 is

$$x_3 = x_2 + \frac{-2c}{b \pm \sqrt{b^2 - 4ac}}$$

$+$ when $|b + D| > |b - D|$

$-$ when $|b + D| < |b - D|$

A table can be drafted to summarize the process such as

k	x_0	x_1	x_2	x_3

The formula for calculating the error e is

$$e = \left| \frac{x_3 - x_2}{x_3} \right| \times 100$$

It can further be noted that when the discriminant is negative, the approximation turns into an imaginary number.

In general, the aforesaid method executes as follows

1) Identify the initial values x_0, x_1 and x_2
2) Solve for the curve-fitting values δ_0 and δ_1, and h_0 and h_1
3) Solve for the coefficients a, b, and c
4) Solve for x_3
5) Substitute x_1 to x_0, x_2 to x_1, and x_3 to x_2
6) Solve for e
7) Repeat

DISCUSSION

MATRIX DECOMPOSITION TECHNIQUE
CHOLESKY'S METHOD

The *Cholesky's Method* or *Cholesky Decomposition* is a numerical solution for system of linear equations by solving for the decomposition $A = LU$, then solving $LR = C$ for R by forward substitution, and solving $UX = R$ for X by backward substitution.

The matrices L and U are

MATRIX L

$$\begin{vmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{vmatrix}$$

MATRIX U

$$\begin{vmatrix} 1 & U_{12} & U_{13} \\ 0 & 1 & U_{23} \\ 0 & 0 & 1 \end{vmatrix}$$

The formulas for the individual values are

$$L_{11} = A_{11} \qquad L_{21} = A_{21} \qquad L_{31} = A_{31}$$

$$U_{12} = \frac{A_{12}}{L_{11}} \qquad U_{13} = \frac{A_{13}}{L_{11}}$$

$$L_{22} = A_{22} - L_{21} \times U_{12} \qquad L_{32} = A_{32} - L_{31} \times U_{12}$$

$$U_{23} = \frac{A_{23} - L_{21} * U_{13}}{L_{22}}$$

$$L_{33} = A_{33} - L_{31} \times U_{13} - L_{32} \times U_{23}$$

In general, the aforesaid method executes as follows

1) Solve for the decomposition $A = LU$
2) Solve for R in $LR = C$ by forward substitution
3) Solve for X in $UX = R$ by backward substitution

.

DISCUSSION

ITERATIVE TECHNIQUE FOR SYSTEM OF LINEAR EQUATIONS
GAUSS-JACOBI METHOD

The *Gauss-Jacobi Method* is an iterative numerical method for approximating solutions of a diagonally dominant system of linear equations.

It begins with three (3) initial values x_1, x_2, and x_3 of zeros that will be substituted aptly to the iterative formula of each of the equation. The results x_1', x_2', and x_3' will be the substituted as next approximations for its corresponding dominant variables x_1, x_2, and x_3.

The iterative formula can be derived by equating the function to its dominant variable. And thus, it is important to arrange the equations such that its dominant variable is located in a diagonal.

The aforesaid process of calculation and substitution will repeat itself until the defined error tolerance e has been satisfied by the approximations.

A table can be drafted to summarize the process such as

k	x_1'	x_2'	x_3'

The formula for calculating the error e is

$$e = |x_i' - x_i|$$

where x_i' is the latest calculated value and x_i the older.

In general, the aforesaid method executes as follows

1) Arrange the equations in diagonal order
2) Derive the iterative formula for each equation
3) Set the initial values x_1, x_2, and x_3 to zero
4) Solve for the new values x_1', x_2', and x_3'
5) Solve for e of each variable
6) Repeat

USER MANUAL

POLYNOMIAL TECHNIQUE
MULLER'S METHOD

The steps for successful utilization of the program are as follows

1) Open MP3

    The program opens with an introductory window containing the button that will open the selected method.

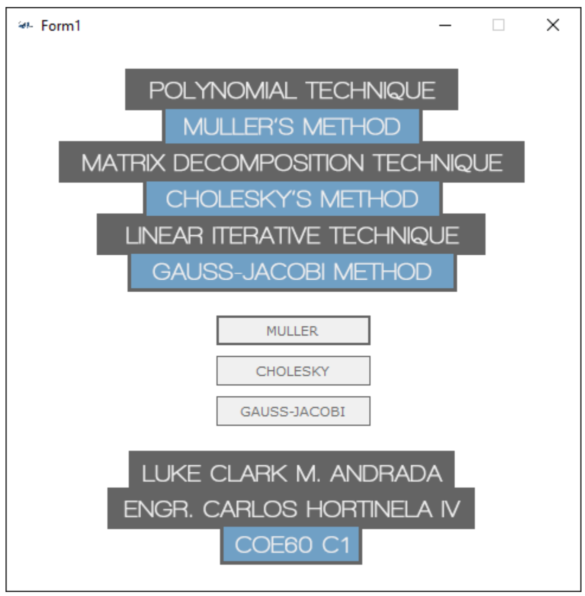

FIG 1. MENU WINDOW

2) Click MULLER

    The button opens another window containing the input and output elements of the selected method.

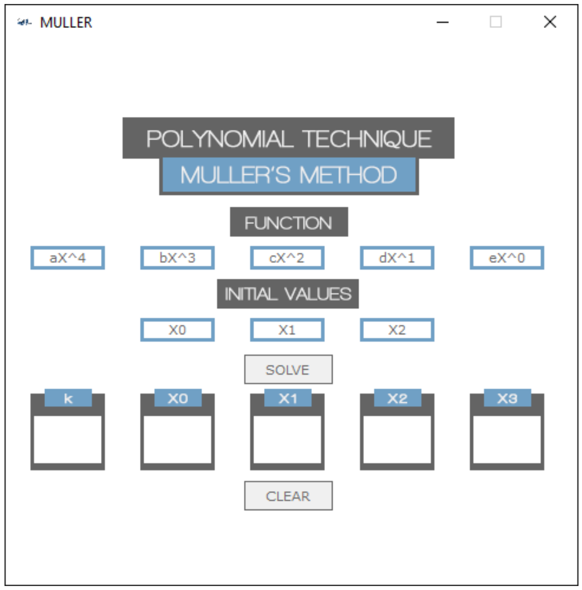

FIG 2. MULLER WINDOW

3) Enter FUNCTION

The program has the capability to accept up to 4th order polynomial but the degree of the polynomial can be changed by setting the coefficient of a variable to zero.

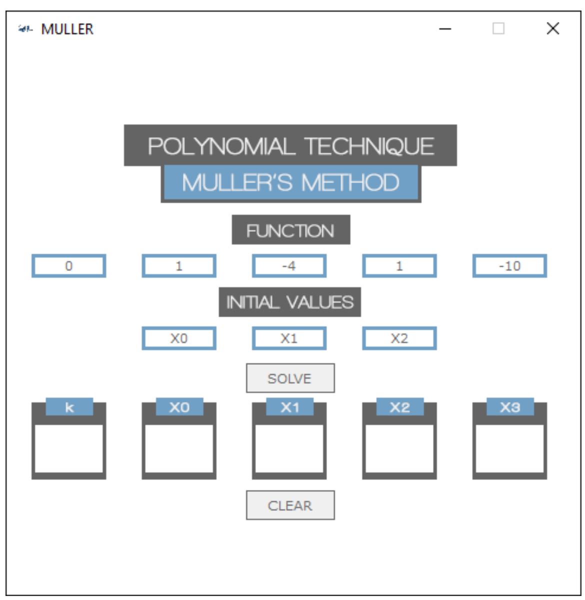

FIG 3. ENTER FUNCTION EX. $x^3 - 4x^2 + x - 10$

4) Enter INITIAL VALUES

The initial values can be identified at the will of the user.

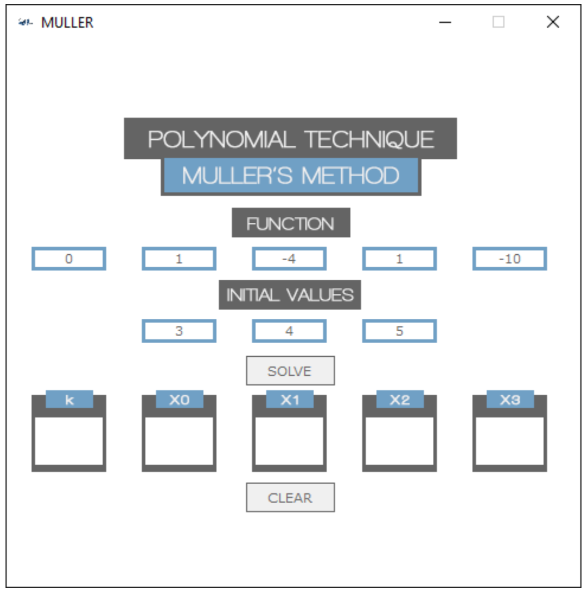

FIG 4. ENTER INITIAL VALUES EX. 3, 4 AND 5

5) Click SOLVE

The program solves for the root of the function implementing the rules of the selected method, drafts the table, and displays the result.

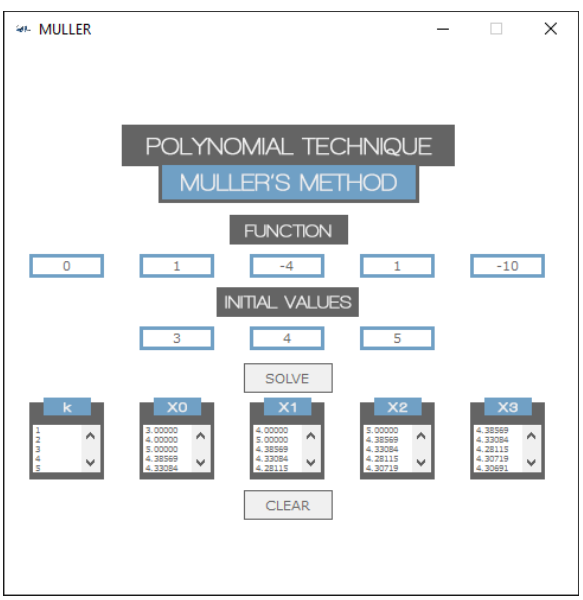

FIG 5. TABLE DRAFT

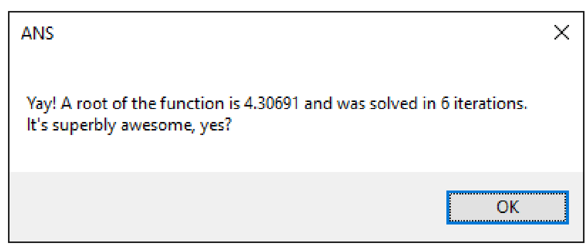

FIG 6. RESULT DISPLAY

6) Click CLEAR

The program resets the window and its elements to start over.

The other specifications of the program are as follows:

- It works in five (5) decimal places and double data type.
- It limits the iteration to one thousand (1000).
- It handles no-input, invalid-input, zero-division, nan-result, and inf-result, and imaginary-result calculation and programming errors.

USER MANUAL

MATRIX DECOMPOSITION
CHOLESKY'S METHOD

The steps for successful utilization of the program are as follows

1) Open MP3

    The program opens with an introductory window containing the button that will open the selected method.

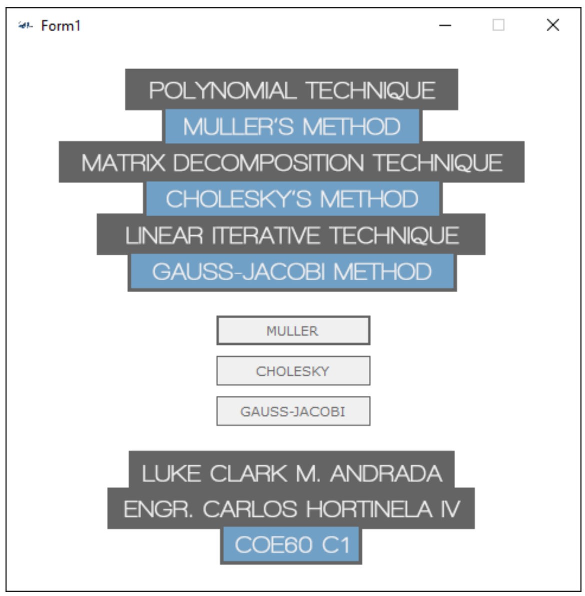

FIG 1. MENU WINDOW

2) Click CHOLESKY

    The button opens another window containing the input and output elements of the selected method.

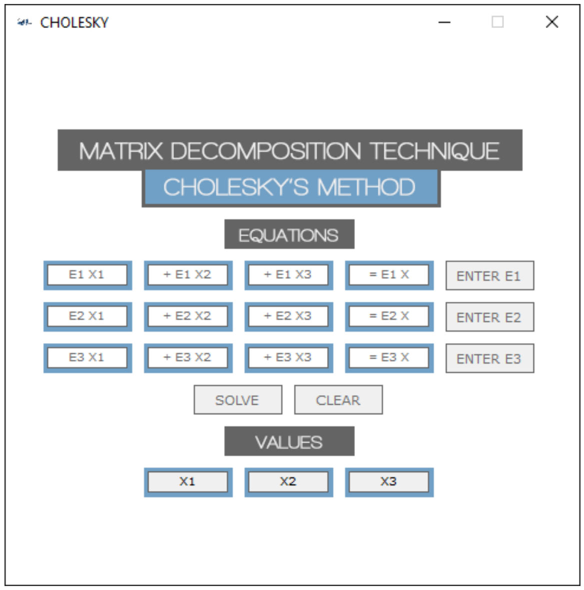

FIG 2. CHOLESKY WINDOW

3) Enter EQUATIONS

Only the coefficients of the variables and the constants are entered.

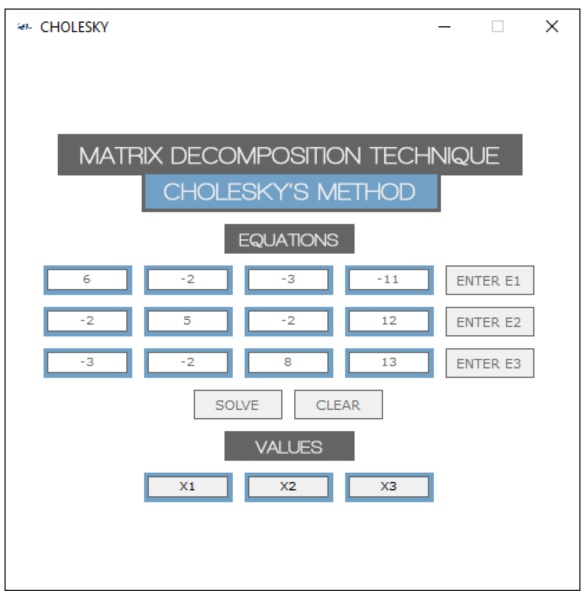

FIG 3. ENTER FUNCTIONS EX. $6x - 2x - 3x = -11$

4) Click ENTER

The program acknowledges the equations.

5) Click SOLVE

The program solves for the solutions of the system of linear equations implementing the rules of the selected method and displays the results.

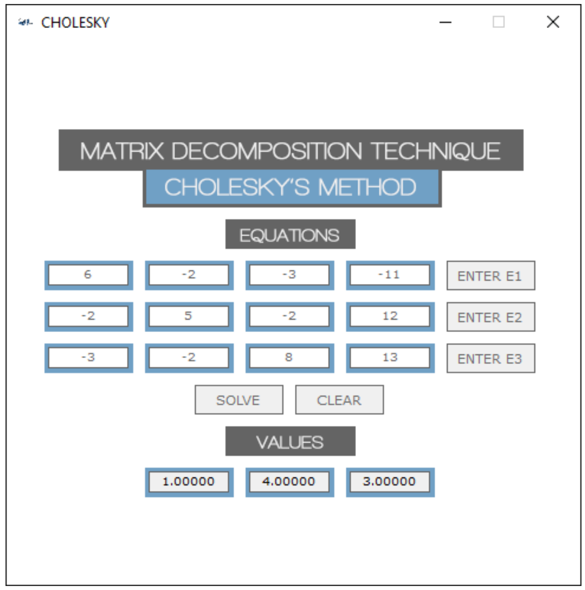

FIG 4. RESULT DISPLAY

6) Click CLEAR

The program resets the window and its elements to start over.

The other specifications of the program are as follows:

- It works in five (5) decimal places and double data type.
- It handles no-input, invalid-input, diagonal-dominant, and equation-enter calculation and programming errors.

USER MANUAL

ITERATIVE TECHNIQUE FOR SYSTEM OF LINEAR EQUATIONS
GAUSS-JACOBI METHOD

The steps for successful utilization of the program are as follows

1) Open MP3

   The program opens with an introductory window containing the button that will open the selected method.

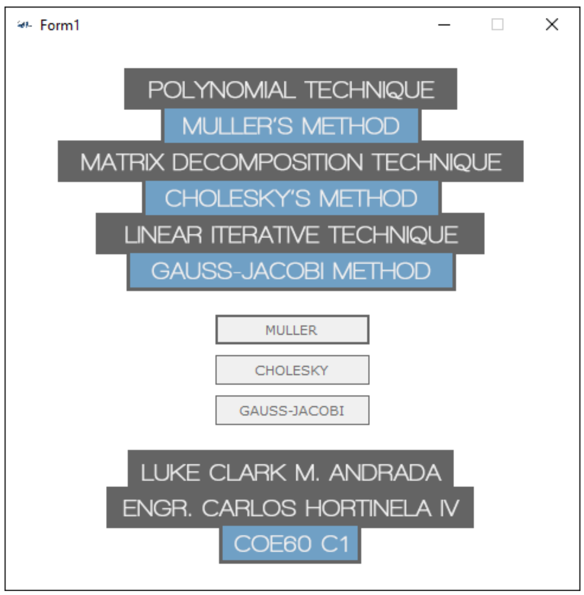

FIG 1. MENU WINDOW

2) Click GAUSS-JACOBI

   The button opens another window containing the input and output elements of the selected method.

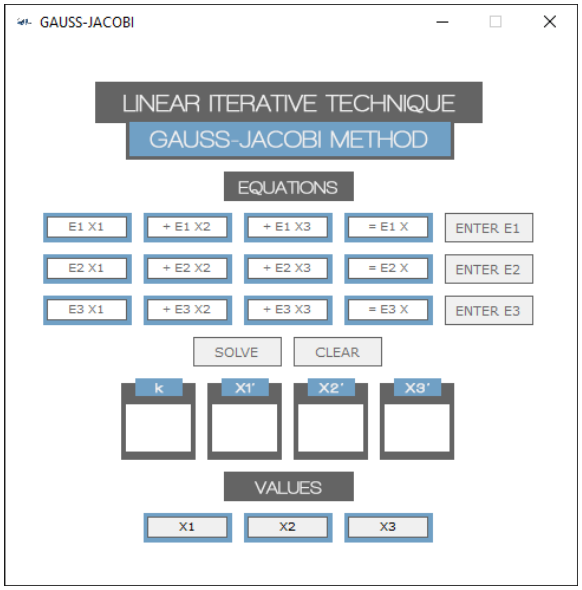

FIG 2. GAUSS-JACOBI WINDOW

3) Enter EQUATIONS

Only the coefficients of the variables and the constants are entered.

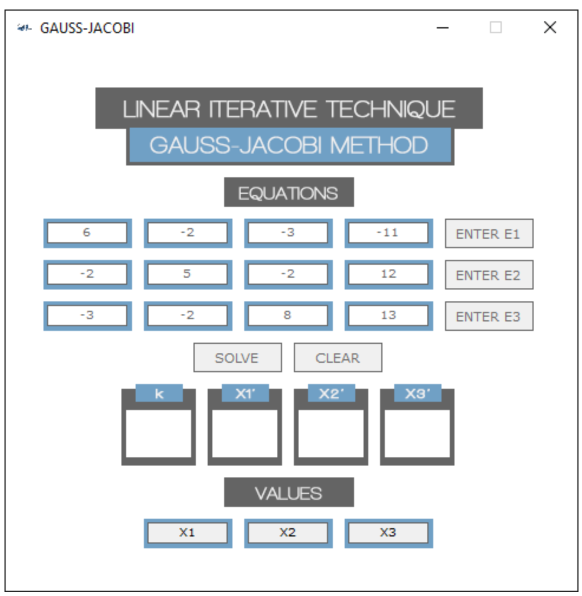

FIG 3. ENTER FUNCTIONS EX. $6x - 2x - 3x = -11$

4) Click ENTER

The program acknowledges the equations.

5) Click SOLVE

The program solves for the solutions of the system of linear equations implementing the rules of the selected method, drafts the table, and displays the results.

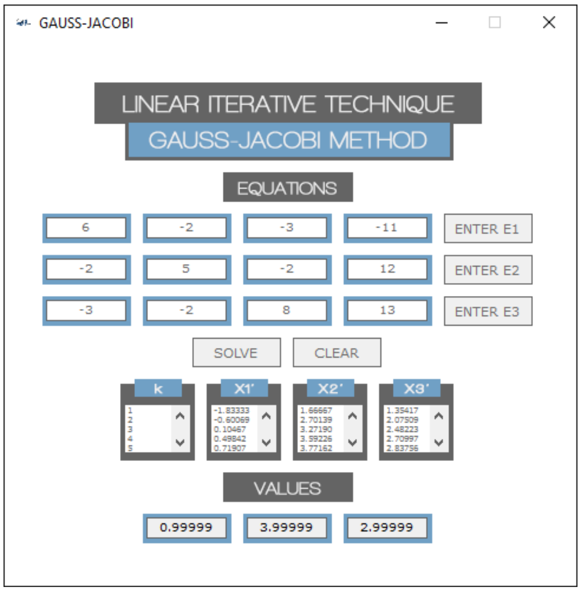

FIG 4. TABLE DRAFT

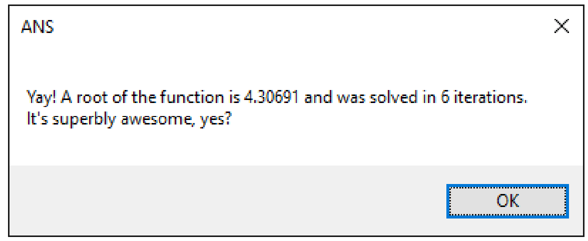

FIG 5. RESULT DISPLAY

6) Click CLEAR

The program resets the window and its elements to start over.

The other specifications of the program are as follows:

- It works in five (5) decimal places and double data type.
- It limits the iteration to one thousand (1000).
- It handles no-input, invalid-input, diagonal-dominant, and equation-enter calculation and programming errors.

APPENDIX

POLYNOMIAL TECHNIQUE
MULLER'S METHOD

SOURCE CODE

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;

/*
   LUKE CLARK M. ANDRADA

   ENGR. CARLOS HORTINELA IV
   COE60 C1

   MP3
   POLYNOMIAL TECHNIQUE x MULLER'S METHOD
   MATRIX DECOMPOSITION TECHNIQUE x CHOLESKY'S METHOD
   LINEAR ITERATIVE TECHNIQUE x GAUSS-JACOBI METHOD

   REFERENCES
   stackoverflow.com
   msdn.microsoft.com
   existing applications
*/

namespace MP3
{
   public partial class frmMain : Form
   {
      public frmMain()
      {
         InitializeComponent();
      }

      private void btnMuller_Click(object sender, EventArgs e)
      {
         MP3.frmMuller form = new MP3.frmMuller();
         form.ShowDialog();
      }

      private void btnCholesky_Click(object sender, EventArgs e)
      {
         MP3.frmCholesky form = new MP3.frmCholesky();
         form.ShowDialog();
      }
```

```
        private void btnGaussJacobi_Click(object sender, EventArgs e)
        {
            MP3.frmGaussJacobi form = new MP3.frmGaussJacobi();
            form.ShowDialog();
        }
    }
}
```

FORM2.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;

namespace MP3
{
    public partial class frmMuller : Form
    {
        public frmMuller()
        {
            InitializeComponent();
        }

        private void btnMuller_solve_Click(object sender, EventArgs e)
        {
            // initialize variables
            double s1, s0, h1, h0, a, b, c, d, f, x1, x0, x2, x3, f0, f1, f2, error = 9999, count = 0, a1, b1, c1, d1;
            double[] data = new double[5];

            // handle errors
            try
            {
                x0 = double.Parse(txtBoxMuller_x0.Text);
                x1 = double.Parse(txtBoxMuller_x1.Text);
                x2 = double.Parse(txtBoxMuller_x2.Text);

                a = double.Parse(txtBoxMuller_a.Text);
                b = double.Parse(txtBoxMuller_b.Text);
                c = double.Parse(txtBoxMuller_c.Text);
                d = double.Parse(txtBoxMuller_d.Text);
                f = double.Parse(txtBoxMuller_e.Text);

                if (a == 0 && b == 0 && c == 0 && d == 0 && f == 0)
                {
                    clearBox();
                    MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR",
MessageBoxButtons.OK);
                    return;
                }
            }

            catch (FormatException)
```

```
            {
               clearBox();
               MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR", MessageBoxButtons.OK);
               return;
            }

            // clear boxes
            clearBox();

            while (error > .00001 && count < 1000)
            {
               // solve for the values of y
               f0 = (a * (x0 * x0 * x0 * x0)) + (b * x0 * x0 * x0) + (c * x0 * x0) + (d * x0) + f;
               f1 = (a * (x1 * x1 * x1 * x1)) + (b * x1 * x1 * x1) + (c * x1 * x1) + (d * x1) + f;
               f2 = (a * (x2 * x2 * x2 * x2)) + (b * x2 * x2 * x2) + (c * x2 * x2) + (d * x2) + f;

               // solve for the curve-fitting values
               h0 = x1 - x0;
               h1 = x2 - x1;
               s0 = (f1 - f0) / h0;
               s1 = (f2 - f1) / h1;

               // solve for the coefficients
               a1 = (s1 - s0) / h1 + h0;
               b1 = (a1 * h1) + s1;
               c1 = f2;

               // solve for the base discriminant
               d1 = (b1 * b1) - (4 * a1 * c1);

               // check if the base discriminant is positive
               if (d1 >= 0 && b1 - d1 != 0)
               {
                  // solve for x3
                  d1 = Math.Sqrt(d1);
                  if (Math.Abs(b1 + d1) > Math.Abs(b1 - d1))
                     x3 = x2 + ((-2 * c1) / (b1 + d1));
                  else
                     x3 = x2 + ((-2 * c1) / (b1 - d1));

                  // solve for error
                  error = Math.Abs((x3 - x2));

                  // iteration counter
                  count++;

                  // write values
                  lBoxMuller_k.Items.Add(count);
                  lBoxMuller_x0.Items.Add(Math.Round(x0, 5).ToString("0.00000"));
                  lBoxMuller_x1.Items.Add(Math.Round(x1, 5).ToString("0.00000"));
                  lBoxMuller_x2.Items.Add(Math.Round(x2, 5).ToString("0.00000"));
                  lBoxMuller_x3.Items.Add(Math.Round(x3, 5).ToString("0.00000"));

                  // swap values
                  x0 = x1;
                  x1 = x2;
                  x2 = x3;
```

```
            } else
            {
                // clear boxes
                clearBox();

                // promt user if the result is imaginary
                MessageBox.Show("Hi. I'm sorry but with the given function and initial values, I can tell you that the result will be imaginary and I can't handle imaginary.\nI want you to try again, okay?", "ERR", MessageBoxButtons.OK);
                error = 9999;
                return;
            }
        };

        if (count == 1000)
        {
            clearBox();
            MessageBox.Show("Hi. I'm sorry but I can't solve it.\nI tried but the iterations were over a thousand!", "ERR", MessageBoxButtons.OK);
            return;
        }

        // display result
        MessageBox.Show("Yay! A root of the function is " + Math.Round(x2, 5).ToString("0.00000") + " and was solved in " + count + " iterations.\nIt's superbly awesome, yes?", "ANS", MessageBoxButtons.OK);
    }

    private void btnMuller_clear_Click(object sender, EventArgs e)
    {
        // clear everything
        txtBoxMuller_e.Text = "eX^0";
        txtBoxMuller_d.Text = "dX^1";
        txtBoxMuller_c.Text = "cX^2";
        txtBoxMuller_b.Text = "bX^3";
        txtBoxMuller_a.Text = "aX^4";
        txtBoxMuller_x0.Text = "X0";
        txtBoxMuller_x1.Text = "X1";
        txtBoxMuller_x2.Text = "X2";

        clearBox();
    }

    private void clearBox()
    {
        lBoxMuller_k.Items.Clear();
        lBoxMuller_x0.Items.Clear();
        lBoxMuller_x1.Items.Clear();
        lBoxMuller_x2.Items.Clear();
        lBoxMuller_x3.Items.Clear();
    }
  }
}
```

APPENDIX

MATRIX DECOMPOSITION METHOD
CHOLESKY'S METHOD

SOURCE CODE

FORM3.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;

namespace MP3
{
   public partial class frmCholesky : Form
   {
      // globally initialize variables
      double y1, y2, y3, x1, x2, x3;
      double[,] A = new double[3, 4];
      double[,] L = new double[3, 3];
      double[,] U = new double[3, 3];
      bool flag1 = false, flag2 = false, flag3 = false;

      public frmCholesky()
      {
         InitializeComponent();
      }

      private void frmCholesky_Load(object sender, EventArgs e)
      {
         flag1 = false;

         // initialize matrices
         for (int i = 0; i < 3; i++)
         {
            for (int j = 0; j < 3; j++)
            {
               L[i, j] = 0;
               U[i, j] = 0;
            }
         }

         U[0, 0] = 1;
         U[1, 1] = 1;
         U[2, 2] = 1;
      }

      private void btnCholesky_e1_Click(object sender, EventArgs e)
      {
```

```
            // initialize input and handle errors
            try
            {
                A[0, 0] = double.Parse(txtBoxCholesky_e1x1.Text);
                A[0, 1] = double.Parse(txtBoxCholesky_e1x2.Text);
                A[0, 2] = double.Parse(txtBoxCholesky_e1x3.Text);
                A[0, 3] = double.Parse(txtBoxCholesky_e1x.Text);
            }

            catch (FormatException)
            {
                clearBox();
                MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR", 
MessageBoxButtons.OK);
                flag1 = false;
                return;
            }

            // check if correctly arranged to diagonal formation
            if (A[0, 0] >= A[0, 1] && A[0, 0] >= A[0, 2])
            {
                flag1 = true;
                return;
            }

            // clear first equation otherwise
            clearBox();
            MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nThe first coefficient should be the largest.\nI want 
you to try again, okay?", "ERR", MessageBoxButtons.OK);
            flag1 = false;
        }

        private void btnCholesky_e2_Click(object sender, EventArgs e)
        {
            // initialize input and handle errors
            try
            {
                A[1, 0] = double.Parse(txtBoxCholesky_e2x1.Text);
                A[1, 1] = double.Parse(txtBoxCholesky_e2x2.Text);
                A[1, 2] = double.Parse(txtBoxCholesky_e2x3.Text);
                A[1, 3] = double.Parse(txtBoxCholesky_e2x.Text);
            }

            catch (FormatException)
            {
                clearBox();
                MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR", 
MessageBoxButtons.OK);
                flag2 = false;
                return;
            }

            // check if correctly arranged to diagonal structure
            if (A[1, 1] >= A[1, 0] && A[1, 1] >= A[1, 2])
            {
                flag2 = true;
                return;
            }
```

```
            // clear second equation otherwise
            clearBox();
            MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nThe second coefficient should be the largest.\nI want
you to try again, okay?", "ERR", MessageBoxButtons.OK);
            flag2 = false;
        }

        private void btnCholesky_e3_Click(object sender, EventArgs e)
        {
            // initialize input and handle errors
            try
            {
                A[2, 0] = double.Parse(txtBoxCholesky_e3x1.Text);
                A[2, 1] = double.Parse(txtBoxCholesky_e3x2.Text);
                A[2, 2] = double.Parse(txtBoxCholesky_e3x3.Text);
                A[2, 3] = double.Parse(txtBoxCholesky_e3x.Text);
            }

            catch (FormatException)
            {
                clearBox();
                MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR",
MessageBoxButtons.OK);
                flag3 = false;
                return;
            }

            // check if correctly arranged to diagonal structure
            if (A[2, 2] >= A[2, 0] && A[2, 2] >= A[2, 1])
            {
                flag3 = true;
                return;
            }

            // clear second equation otherwise
            clearBox();
            MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nThe third coefficient should be the largest.\nI want
you to try again, okay?", "ERR", MessageBoxButtons.OK);
            flag3 = false;
        }

        private void btnCholesky_solve_Click(object sender, EventArgs e)
        {
            // check if all equations were entered
            if (!flag1 || !flag2 || !flag3)
            {
                clearBox();
                MessageBox.Show("Hi. A favor, please enter the equations first. Tenk!", "ERR", MessageBoxButtons.OK);
                return;
            }

            // fill the matrices with corresponding values
            for (int i = 0; i < 3; i++)
                L[i, 0] = A[i, 0];

            for (int i = 1; i < 3; i++)
                U[0, i] = A[0, i] / L[0, 0];
```

```
            for (int i = 1; i < 3; i++)
                L[i, 1] = A[i, 1] - L[i, 0] * U[0, 1];

            U[1, 2] = (A[1, 2] - L[1, 0] * U[0, 2]) / L[1, 1];
            L[2, 2] = A[2, 2] - L[2, 0] * U[0, 2] - L[2, 1] * U[1, 2];

            // solve for rs
            y1 = A[0, 3] / L[0, 0];
            y2 = (A[1, 3] + -1 * L[1, 0] * y1) / L[1, 1];
            y3 = (-1 * y1 * L[2, 0] + -1 * L[2, 1] * y2 + A[2, 3]) / L[2, 2];

            // solve for xs
            x3 = y3;
            x2 = y2 + x3 * -1 * U[1, 2];
            x1 = y1 + -1 * U[0, 1] * x2 + -1 * U[0, 2] * x3;

            // write the results
            txtBoxCholesky_x1.Text = Convert.ToString(Math.Round(x1, 5).ToString("0.00000"));
            txtBoxCholesky_x2.Text = Convert.ToString(Math.Round(x2, 5).ToString("0.00000"));
            txtBoxCholesky_x3.Text = Convert.ToString(Math.Round(x3, 5).ToString("0.00000"));

            MessageBox.Show("Yay! I don't have anything to say but,\nIt's superbly awesome, yes?", "ANS", MessageBoxButtons.OK);
        }

        private void btnCholesky_clear_Click(object sender, EventArgs e)
        {
            // clear everything
            txtBoxCholesky_e1x1.Text = "E1 X1";
            txtBoxCholesky_e1x2.Text = "+ E1 X2";
            txtBoxCholesky_e1x3.Text = "+ E1 X3";
            txtBoxCholesky_e1x.Text = "= E1 X";

            txtBoxCholesky_e2x1.Text = "E2 X1";
            txtBoxCholesky_e2x2.Text = "+ E2 X2";
            txtBoxCholesky_e2x3.Text = "+ E2 X3";
            txtBoxCholesky_e2x.Text = "= E2 X";

            txtBoxCholesky_e3x1.Text = "E3 X1";
            txtBoxCholesky_e3x2.Text = "+ E3 X2";
            txtBoxCholesky_e3x3.Text = "+ E3 X3";
            txtBoxCholesky_e3x.Text = "= E3 X";

            clearBox();

            flag1 = flag2 = flag3 = false;
        }

        private void clearBox()
        {
            txtBoxCholesky_x1.Text = "X1";
            txtBoxCholesky_x2.Text = "X2";
            txtBoxCholesky_x3.Text = "X3";
        }
    }
}
```

APPENDIX

ITERATIVE TECHNIQUES FOR SYSTEM OF LINEAR EQUATIONS
GAUSS-JACOBI METHOD

SOURCE CODE

FORM4.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;

namespace MP3
{
  public partial class frmGaussJacobi : Form
  {
    // globally initialize variables
    double[] first = new double[10];
    double[] second = new double[10];
    double[] third = new double[10];
    double error = 0.00001;
    double v1, v2, v3, temp1, temp2, temp3, tempf1, tempf2, tempf3;
    int count = 0;
    bool flag1 = false, flag2 = false, flag3 = false;

    public frmGaussJacobi()
    {
      InitializeComponent();
    }

    private void btnGaussJacobi_e1_Click(object sender, EventArgs e)
    {
      // initialize input and handle errors
      try
      {
        first[0] = double.Parse(txtBoxGaussJacobi_e1x1.Text);
        first[1] = double.Parse(txtBoxGaussJacobi_e1x2.Text);
        first[2] = double.Parse(txtBoxGaussJacobi_e1x3.Text);
        first[3] = double.Parse(txtBoxGaussJacobi_e1x.Text);
      }

      catch (FormatException)
      {
        clearBox();
        MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR",
MessageBoxButtons.OK);
        flag1 = false;
        return;
      }
```

```
            // check if correctly arranged to diagonal structure
            if (first[0] >= first[1] && first[0] >= first[2])
            {
                flag1 = true;
                return;
            }

            // clear first equation otherwise
            clearBox();
            MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nThe first coefficient should be the largest.\nI want
you to try again, okay?", "ERR", MessageBoxButtons.OK);
            flag1 = false;
        }

        private void btnGaussJacobi_e2_Click(object sender, EventArgs e)
        {
            // initialize input and handle errors
            try
            {
                second[0] = double.Parse(txtBoxGaussJacobi_e2x1.Text);
                second[1] = double.Parse(txtBoxGaussJacobi_e2x2.Text);
                second[2] = double.Parse(txtBoxGaussJacobi_e2x3.Text);
                second[3] = double.Parse(txtBoxGaussJacobi_e2x.Text);
            }

            catch (FormatException)
            {
                clearBox();
                MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR",
MessageBoxButtons.OK);
                flag2 = false;
                return;
            }

            // check if correctly arranged to diagonal structure
            if (second[1] >= second[0] && second[1] >= second[2])
            {
                flag2 = true;
                return;
            }

            // clear second equation otherwise
            clearBox();
            MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nThe second coefficient should be the largest.\nI want
you to try again, okay?", "ERR", MessageBoxButtons.OK);
            flag2 = false;
        }

        private void btnGaussJacobi_e3_Click(object sender, EventArgs e)
        {
            // initialize input and handle errors
            try
            {
                third[0] = double.Parse(txtBoxGaussJacobi_e3x1.Text);
                third[1] = double.Parse(txtBoxGaussJacobi_e3x2.Text);
                third[2] = double.Parse(txtBoxGaussJacobi_e3x3.Text);
                third[3] = double.Parse(txtBoxGaussJacobi_e3x.Text);
```

```
        }

        catch (FormatException)
        {
            clearBox();
            MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR",
MessageBoxButtons.OK);
            flag3 = false;
            return;
        }

        // check if correctly arranged to diagonal structure
        if (third[2] >= third[0] && third[2] >= third[1])
        {
            flag3 = true;
            return;
        }

        // clear third equation otherwise
        clearBox();
        MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nThe third coefficient should be the largest.\nI want
you to try again, okay?", "ERR", MessageBoxButtons.OK);
        flag3 = false;
    }

    private void btnGaussJacobi_solve_Click(object sender, EventArgs e)
    {
        clearBox();

        // check if all equations were entered
        if (!flag1 || !flag2 || !flag3)
        {
            MessageBox.Show("Hi. A favor, please enter the equations first. Tenk!", "ERR", MessageBoxButtons.OK);
            return;
        }

        // clear counter
        count = 0;

        // initialize variables
        v1 = v2 = v3 = 0;

        do
        {
            // save v temporarily
            temp1 = v1;
            temp2 = v2;
            temp3 = v3;

            // iteration counter
            count++;

            // solve for the new v
            v1 = (first[3] + (-1 * first[1] * v2) + (-1 * first[2] * v3)) / first[0];
            v2 = (second[3] + (-1 * second[0] * v1) + (-1 * second[2] * v3)) / second[1];
            v3 = (third[3] + (-1 * third[0] * v1) + (-1 * third[1] * v2)) / third[2];

            // write the new v
```

```
            lBoxGaussJacobi_k.Items.Add(count);
            lBoxGaussJacobi_v1.Items.Add(Math.Round(v1, 5).ToString("0.00000"));
            lBoxGaussJacobi_v2.Items.Add(Math.Round(v2, 5).ToString("0.00000"));
            lBoxGaussJacobi_v3.Items.Add(Math.Round(v3, 5).ToString("0.00000"));

            // check error
            tempf1 = Math.Abs(temp1 - v1);
            tempf2 = Math.Abs(temp2 - v2);
            tempf3 = Math.Abs(temp3 - v3);
          } while (tempf1 > error && tempf2 > error && tempf3 > error && count < 1000);

          if (count == 1000)
          {
            clearBox();
            MessageBox.Show("Hi. I'm sorry but I can't solve it.\nI tried but the iterations were over a thousand!", "ERR",
MessageBoxButtons.OK);
            return;
          }

          txtBoxGaussJacobi_x1.Text = Math.Round(v1, 5).ToString("0.00000");
          txtBoxGaussJacobi_x2.Text = Math.Round(v2, 5).ToString("0.00000");
          txtBoxGaussJacobi_x3.Text = Math.Round(v3, 5).ToString("0.00000");

          MessageBox.Show("Yay! The values were solved in " + count + " iterations.\nIt's superbly awesome, yes?", "ANS",
MessageBoxButtons.OK);
        }

        private void btnGaussJacobi_clear_Click(object sender, EventArgs e)
        {
          // clear everything
          txtBoxGaussJacobi_e1x1.Text = "E1 X1";
          txtBoxGaussJacobi_e1x2.Text = "+ E1 X2";
          txtBoxGaussJacobi_e1x3.Text = "+ E1 X3";
          txtBoxGaussJacobi_e1x.Text = "= E1 X";

          txtBoxGaussJacobi_e2x1.Text = "E2 X1";
          txtBoxGaussJacobi_e2x2.Text = "+ E2 X2";
          txtBoxGaussJacobi_e2x3.Text = "+ E2 X3";
          txtBoxGaussJacobi_e2x.Text = "= E2 X";

          txtBoxGaussJacobi_e3x1.Text = "E3 X1";
          txtBoxGaussJacobi_e3x2.Text = "+ E3 X2";
          txtBoxGaussJacobi_e3x3.Text = "+ E3 X3";
          txtBoxGaussJacobi_e3x.Text = "= E3 X";

          clearBox();

          flag1 = flag2 = flag3 = false;
        }

        private void clearBox()
        {
          txtBoxGaussJacobi_x1.Text = "X1";
          txtBoxGaussJacobi_x2.Text = "X2";
          txtBoxGaussJacobi_x3.Text = "X3";

          lBoxGaussJacobi_k.Items.Clear();
          lBoxGaussJacobi_v1.Items.Clear();
```

```
            lBoxGaussJacobi_v2.Items.Clear();
            lBoxGaussJacobi_v3.Items.Clear();
        }
    }
}
```

APPENDIX

REFERENCES

stackoverflow.com

msdn.microsoft.com

existing applications

Mapúa Institute of Technology
School of Electrical, Electronics, and Computer Engineering

Machine Problem 4
Numerical Methods

Andrada, Luke Clark M.
COE60 C1

Engr. Carlos Hortinela IV

March 30, 2016

DISCUSSION

REGRESSION TECHNIQUE

LINEAR REGRESSION

The *Linear Regression* is an approach for modeling relationships between variables and can approximate a linear equation $y = a + bx$ to fit a given set of data points.

It begins by solving for the summation of the parameters x_i, y_i, ${x_i}^2$, and $x_i y_i$ specifically in the endeavor of deriving a linear equation. The aforesaid summations will be substituted to the formula of the coefficients a_0 and a_1 of the model linear equation $y = a_0 + a_1 x + e$ that approximates a fit for the data points.

The formula for calculating the value a_1 is

$$a_1 = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{n \sum {x_i}^2 - (\sum x_i)^2}$$

The formula for calculating the value a_0 is

$$a_0 = \bar{y} - a_1 \bar{x}$$

A table can be drafted to summarize the process such as

x_i	y_i	${x_i}^2$	$x_i y_i$

It can further be noted that the resulting linear equation $y = a_0 + a_1 x$ contains the error e and thus is not a highly accurate approximation.

In general, the aforesaid method executes as follows

1) Identify the data points
2) Solve for the summation of x_i, y_i, ${x_i}^2$, and $x_i y_i$
3) Solve for the coefficients a_0 and a_1
4) Derive the linear equation $y = a_0 + a_1 x$

.

DISCUSSION

INTERPOLATION

NEWTON'S DIVIDED DIFFERENCE INTERPOLATING POLYNOMIAL

The *Newton's Divided Difference Interpolating Polynomial* is a numerical method for deriving a polynomial perfectly fitted for a given set of data points.

It begins by arranging the data points from lowest to highest as per conventions and then calculating the coefficients b_n of its model function through divided differences that is $\frac{f(x_{i+1})-f(x_i)}{x_1-x_0}$.

The model function of the aforesaid method is

$$f_n(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + \cdots + b_n(x - x_0)(x - x_1)(x - x_2)\cdots(x - x_{n-1})$$

where $b_0 = f(x_0)$

A table can be drafted to summarize the process such as

i	x_i	$f(x_i)$	$1st$	$2nd$	$3rd$	$\cdots$

In general, the aforesaid method executes as follows

1) Identify and arrange the data points
2) Solve for the coefficients b_n via divided differences
3) Substitute coefficients and values to the model function
4) Simplify the equation

DISCUSSION

NUMERICAL INTEGRATION

TRAPEZOIDAL RULE

The *Trapezoidal Rule* is a numerical method for approximating the definite integral of $\int_a^b f(x)\,dx$ by approximating the region under the curve as trapezoidal and solving for its area.

It begins by identifying how many segments n of trapezoid should the region under the curve within the limits a and b be divided. The step size h will define the intervals and can be calculated by solving for the difference between the limits a and b and dividing it by segments n. The $f(x_i)$ of each interval point is calculated and the trapezoidal rule pattern is applied on each.

The summation of the pattern is solved and substituted to the integral formula to solve for the integral approximation.

The trapezoidal rule pattern is

$$I = \frac{h}{2}\left[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right]$$

The formula for the step size h is

$$h = \frac{b-a}{n}$$

A table can be drafted to summarize the process such as

i	x_i	$f(x_i)$	$pattern$

It can be further observed that as the number of segments n increases, the accuracy of the approximation increases too.

In general, the aforesaid method executes as follows

1) Identify the segments n
2) Solve for the step size h
3) Solve for the $f(x_i)$ of each interval point
4) Apply the trapezoidal rule pattern
5) Solve for the integral approximation I

DISCUSSION

NUMERICAL DIFFERENTIATION

CENTERED FINITE DIVIDED DIFFERENCE

The *Centered Finite Divided Difference* is a numerical method for approximating the derivative $f'(x)$ of a function at a given point x by finite differences.

It begins by identifying the step size h and calculating for both the first backward x_{i-1} and forward step x_{i+1} from the given value x_i and then solving for $f(x_{i-1})$ and $f(x_{i+1})$. The derivative approximation is then solved by

$$D = f'(x) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h}$$

It can be further noted that the aforesaid method is of the truncated version and that as the step size h decreases, the accuracy of the approximation increases.

In general, the aforesaid method executes as follows

1) Identify the step size h
2) Solve for x_{i-1} and x_{i+1}
3) Solve for $f(x_{i-1})$ and $f(x_{i+1})$
4) Solve for the derivative approximation D

USER MANUAL

REGRESSION TECHNIQUE
LINEAR REGRESSION

The steps for successful utilization of the program are as follows

1) Open MP4

The program opens with an introductory window containing the button that will open the selected method.

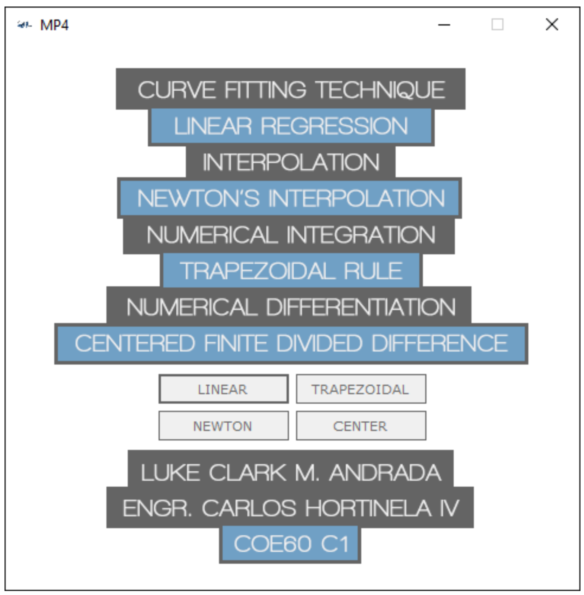

FIG 1. MENU WINDOW

2) Click LINEAR

The button opens another window containing the input and output elements of the selected method.

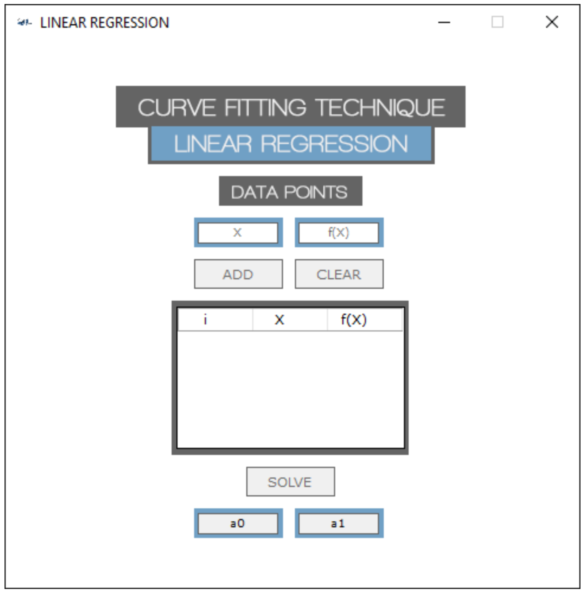

FIG 2. LINEAR REGRESSION WINDOW

3) Enter DATA POINTS

The program has the capability to accept any number of data points.

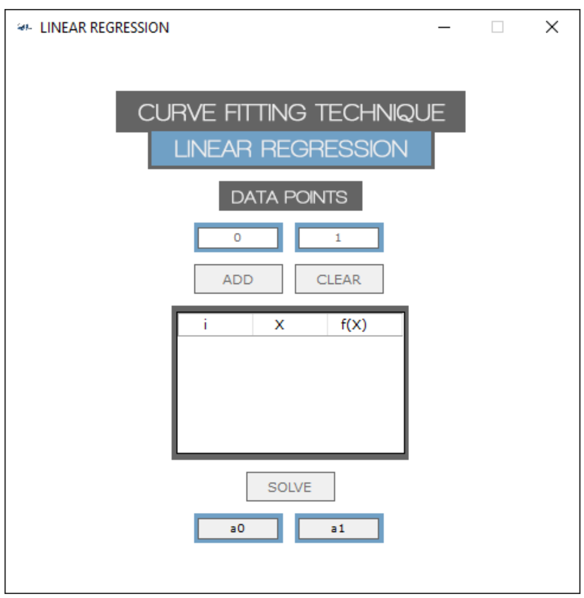

FIG 3. ENTER DATA POINTS EX. $0\ x\ 1$

4) Click ADD

The program acknowledges the data points.

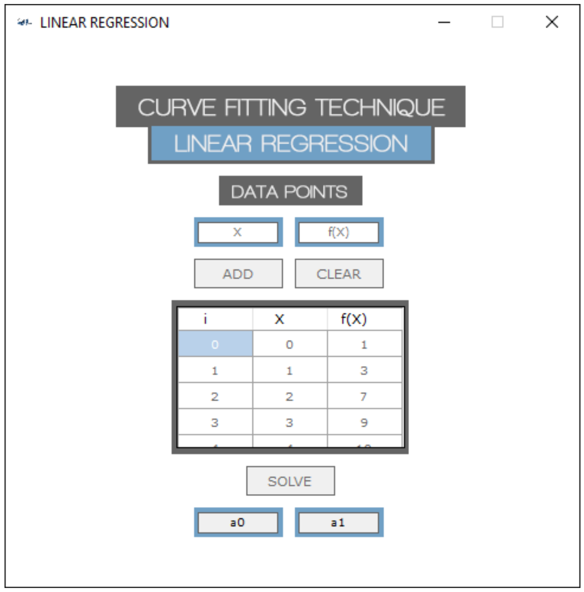

FIG 4. ADD DATA POINTS

5) Click SOLVE

The program solves for the coefficients a_0 and a_1 of the linear model function $y = a_0 + a_1 x$ and displays the results.

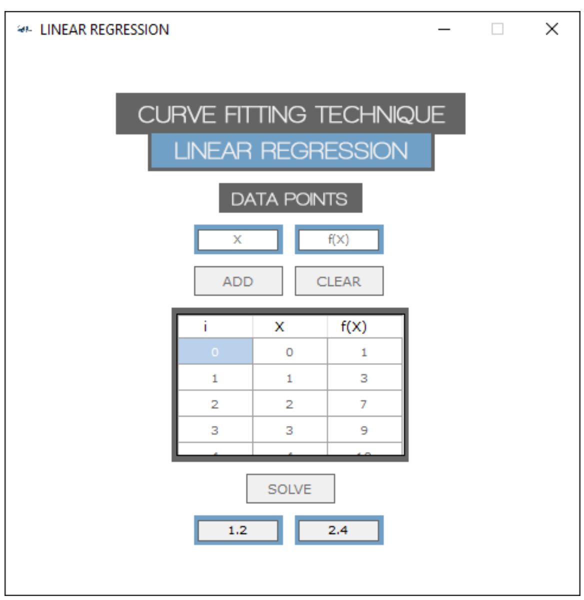

FIG 5. RESULT DISPLAY

6) Click CLEAR

The program resets the window and its elements to start over.

The other specifications of the program are as follows:

- It works in five (5) decimal places and double data type.
- It handles no-input, lack-input, invalid-input, and zero-division calculation and programming errors.

USER MANUAL

INTERPOLATION

NEWTON'S DIVIDED DIFFERENCE INTERPOLATING POLYNOMIAL

The steps for successful utilization of the program are as follows

1) Open MP4

   The program opens with an introductory window containing the button that will open the selected method.

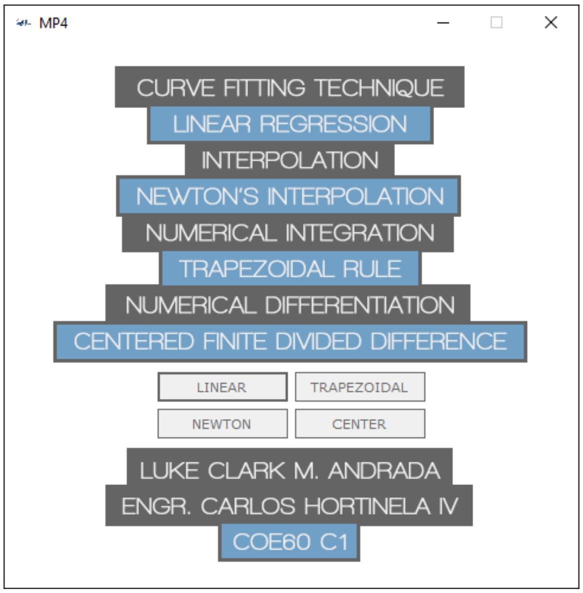

FIG 1. MENU WINDOW

2) Click NEWTON

   The button opens another window containing the input and output elements of the selected method.

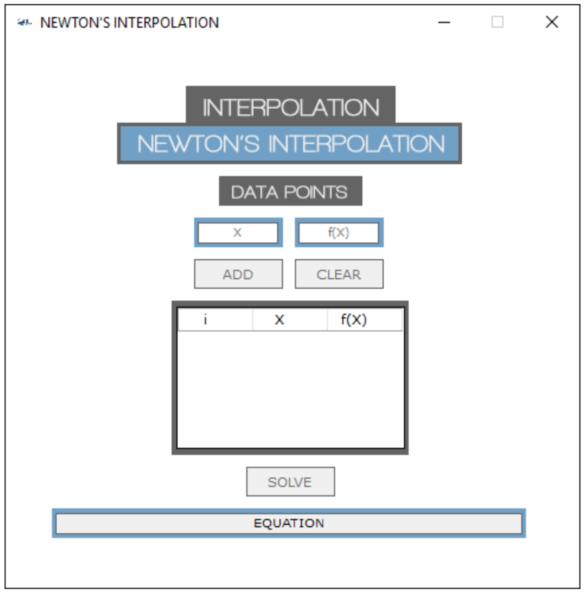

FIG 2. NEWTON'S INTERPOLATION WINDOW

3) Enter DATA POINTS

The program has the capability to accept any number of data points.

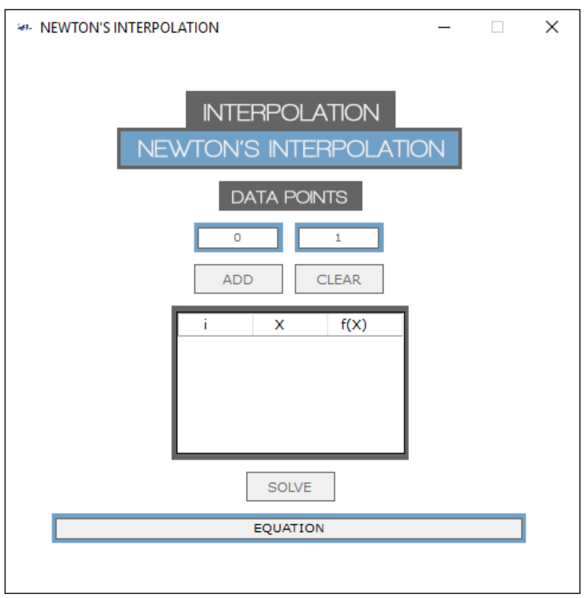

FIG 3. ENTER DATA POINTS EX. $0\ x\ 1$

4) Click ADD

The program acknowledges the data points.

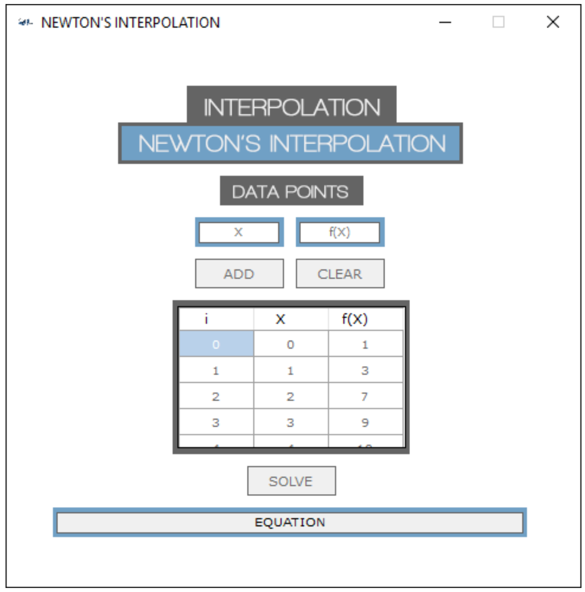

FIG 4. ADD DATA POINTS

5) Click SOLVE

The program solves for the coefficients b_n, substitute the apt values to the model function, and displays the result.

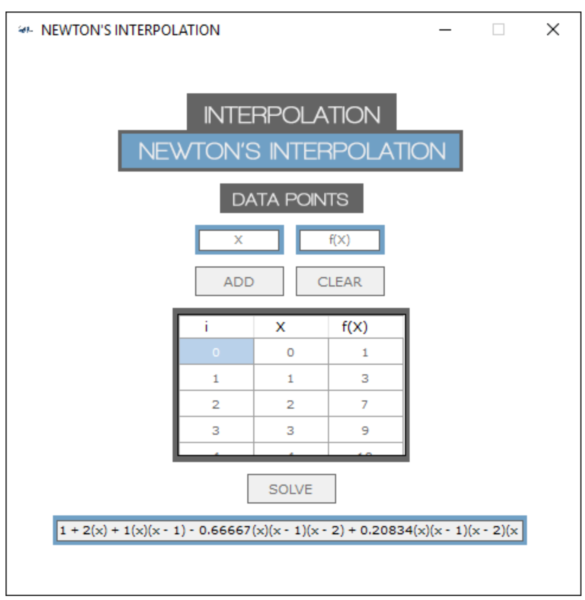

FIG 5. RESULT DISPLAY

6) Click CLEAR

The program resets the window and its elements to start over.

The other specifications of the program are as follows:

- It works in five (5) decimal places and double data type.
- It handles no-input, invalid-input, and zero-division calculation and programming errors.

USER MANUAL

NUMERICAL INTEGRATION
TRAPEZOIDAL RULE PATTERN

The steps for successful utilization of the program are as follows

1) Open MP4

    The program opens with an introductory window containing the button that will open the selected method.

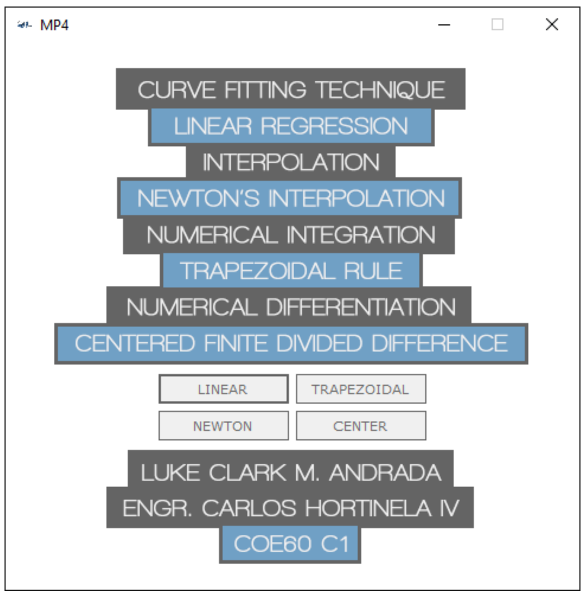

FIG 1. MENU WINDOW

2) Click TRAPEZOIDAL

    The button opens another window containing the input and output elements of the selected method.

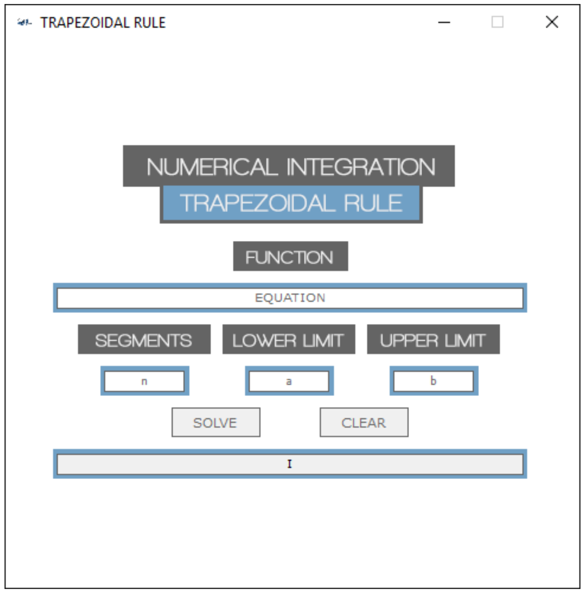

FIG 2. TRAPEZOIDAL RULE WINDOW

3) Enter EQUATION

The program has the capability to accept both polynomials and transcendental functions and further supports several operators, functions, and constants with the help of an external math parser. It can accept and support the following

OPERATORS

+	-	*	/	^	%

FUNCTIONS

SQRT	SIN	COS	TAN	ATAN	ACOS	ASIN	SINH	COSH	TANH	ACOTAN
EXP	LN	LOG	ABS	CIEL	FAC	SFAC	ROUND	FLOOR	FPART	

CONSTANTS

PI	EULER	FALSE	INFINITY

The correct syntax can be consulted at lundin.info/mathparser.

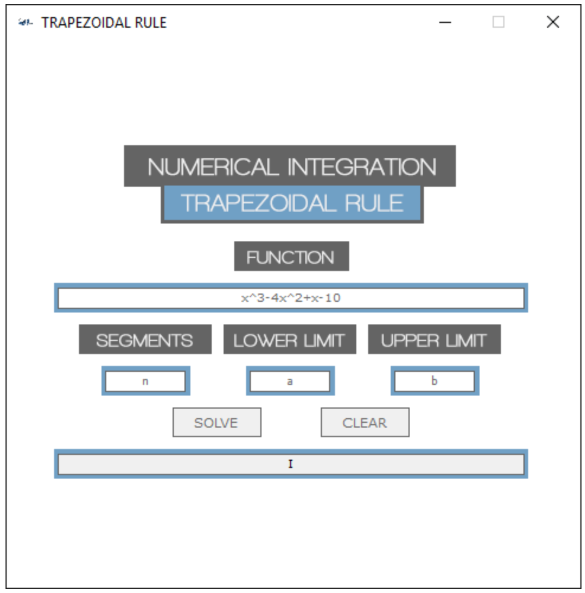

FIG 3. ENTER FUNCTION EX. $x^3 - 4x^2 + x - 10$

4) Enter SEGMENTS, LOWER LIMIT, and UPPER LIMIT

The program has the capability to accept any number of segments except zero and below.

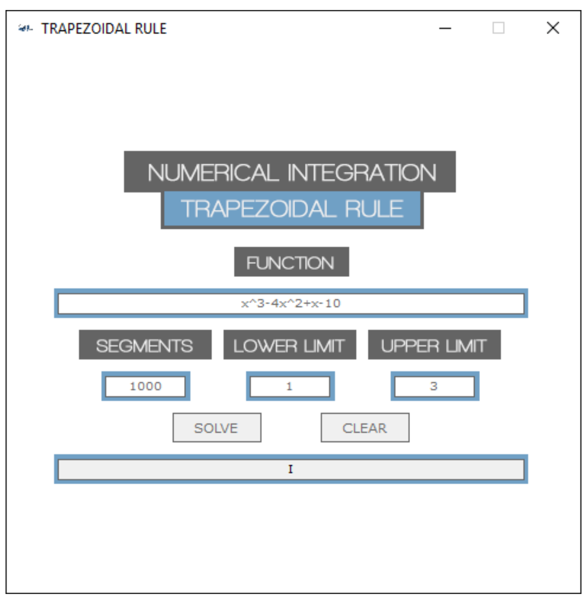

FIG 4. ENTER SEGMENTS, LOWER LIMIT, UPPER LIMIT EX. 1000, 1, 3

5) Click SOLVE

The program solves for the integral approximation at step size h within the limits a and b, and displays the results.

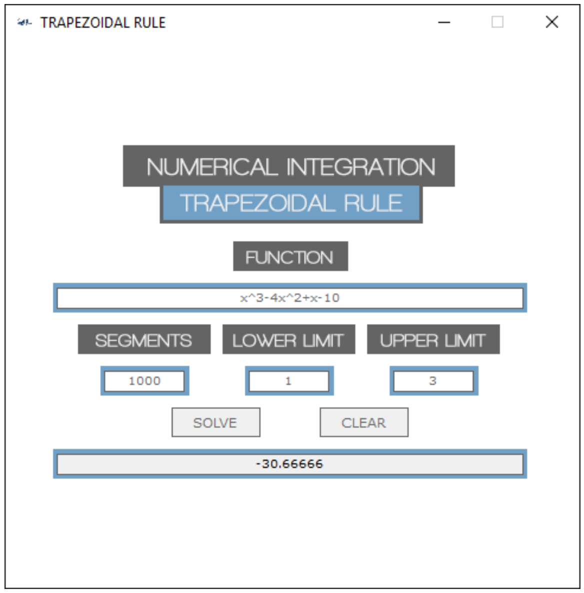

FIG 5. RESULT DISPLAY

6) Click CLEAR

The program resets the window and its elements to start over.

The other specifications of the program are as follows:

- It works in five (5) decimal places and double data type.
- It handles no-input, invalid-input, zero-division, nan-result, and inf-result calculation and programming errors.

USER MANUAL

NUMERICAL DIFFERENTIATION
CENTERED FINITE DIVIDED DIFFERENCE

The steps for successful utilization of the program are as follows

1) Open MP4

The program opens with an introductory window containing the button that will open the selected method.

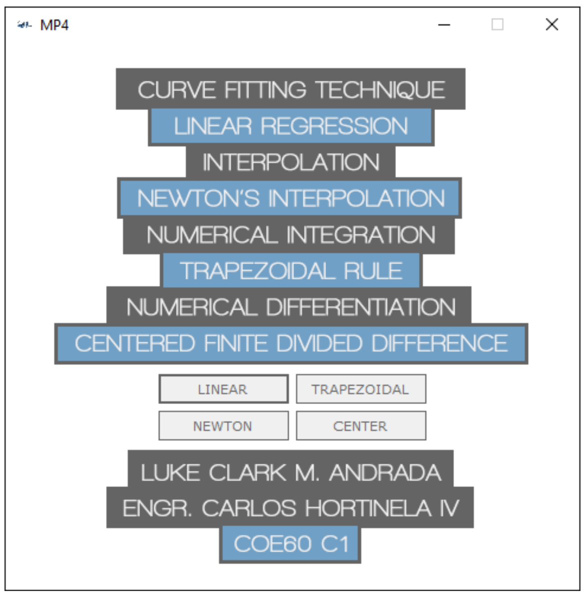

FIG 1. MENU WINDOW

2) Click CENTER

The button opens another window containing the input and output elements of the selected method.

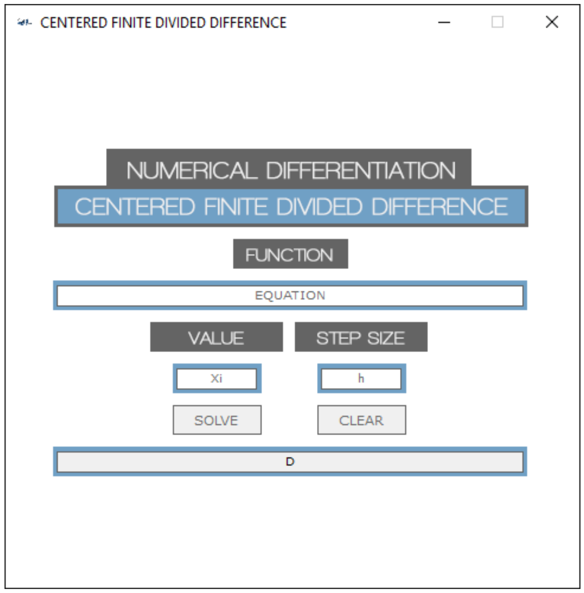

FIG 2. CENTERED FINITE DIVIDED DIFFERENCE WINDOW

3) Enter EQUATION

The program has the capability to accept both polynomials and transcendental functions and further supports several operators, functions, and constants with the help of an external math parser. It can accept and support the following

OPERATORS

+	-	*	/	^	%

FUNCTIONS

SQRT	SIN	COS	TAN	ATAN	ACOS	ASIN	SINH	COSH	TANH	ACOTAN
EXP	LN	LOG	ABS	CIEL	FAC	SFAC	ROUND	FLOOR	FPART	

CONSTANTS

PI	EULER	FALSE	INFINITY

The correct syntax can be consulted at lundin.info/mathparser.

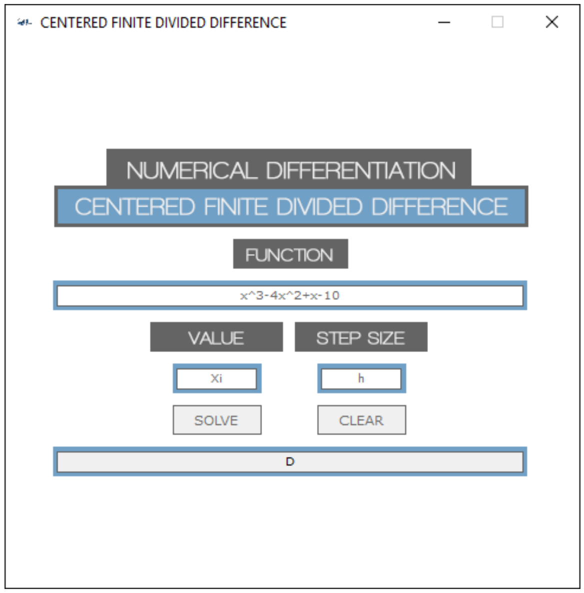

FIG 3. ENTER FUNCTION EX. $x^3 - 4x^2 + x - 10$

4) Enter VALUE and STEP SIZE

The program has the capability to accept any number of step size.

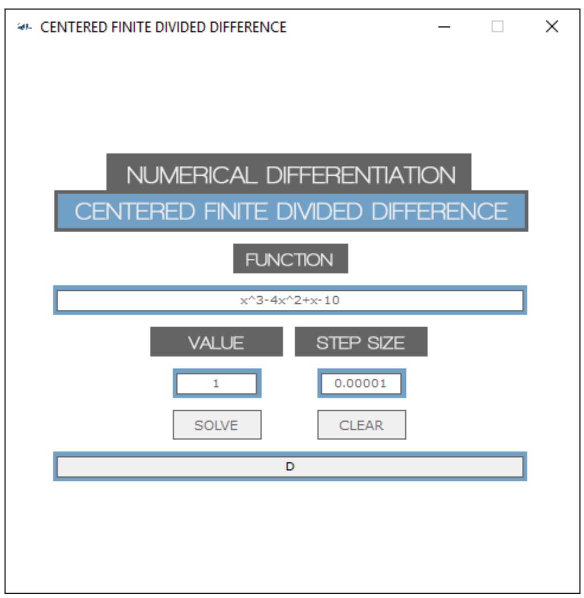

FIG 4. ENTER VALUE, STEP SIZE EX. 1, 0.00001

5) Click SOLVE

The program solves for the derivative approximation at step size h at the given value x_i, and displays the result.

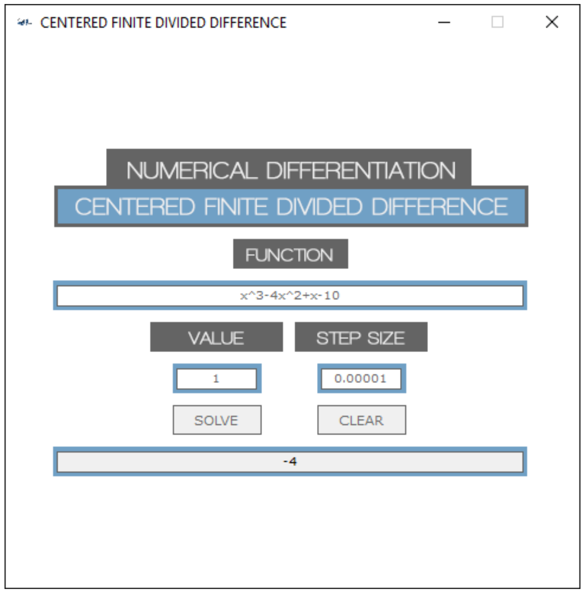

FIG 5. RESULT DISPLAY

6) Click CLEAR

The program resets the window and its elements to start over.

The other specifications of the program are as follows:

- It works in five (5) decimal places and double data type.
- It handles no-input, invalid-input, zero-division, nan-result, and inf-result calculation and programming errors.

APPENDIX

REGRESSION TECHNIQUE
LINEAR REGRESSION

SOURCE CODE

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;

/*
   LUKE CLARK M. ANDRADA

   ENGR. CARLOS HORTINELA IV
   COE60 C1

   MP4
   CURVE FITTING TECHNIQUES x LINEAR REGRESSION
   INTERPOLATION x NEWTON'S DIVIDED DIFFERENCE INTERPOLATING POLYNOMIAL
   NUMERICAL INTEGRATION x TRAPEZOIDAL RULE
   NUMERICAL DIFFERENTIATION x CENTERED FINITE DIVIDED DIFFERENCE

   REFERENCES
   lundin.info/mathparser
   stackoverflow.com
   msdn.microsoft.com
   existing applications
*/

namespace MP4
{
   public partial class frmMain : Form
   {
      public frmMain()
      {
         InitializeComponent();
      }

      private void btnLinear_Click(object sender, EventArgs e)
      {
         MP4.frmLinear form = new MP4.frmLinear();
         form.ShowDialog();
      }

      private void btnNewton_Click(object sender, EventArgs e)
      {
         MP4.frmNewton form = new MP4.frmNewton();
```

```
            form.ShowDialog();
        }

        private void btnTrapezoidal_Click(object sender, EventArgs e)
        {
            MP4.frmTrapezoidal form = new MP4.frmTrapezoidal();
            form.ShowDialog();
        }

        private void btnCentered_Click(object sender, EventArgs e)
        {
            MP4.frmCenter form = new MP4.frmCenter();
            form.ShowDialog();
        }
    }
}
```

FORM2.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;

namespace MP4
{
    public partial class frmLinear : Form
    {
        // globally initialize variables
        int dataLinear_xfxSize = 0;

        int n;
        double sumx, sumy, sumxy;
        double a0, a1;
        double sumxx;
        double temp, check;
        int i;

        public frmLinear()
        {
            InitializeComponent();
        }

        public void solve()
        {
            // reinitialize variables
            n = dataLinear_xfx.Rows.Count;
            sumx = 0.0;
            sumy = 0.0;
            sumxy = 0.0;
            a0 = 0.0;
            a1 = 0.0;
            sumxx = 0.0;
            temp = 0.0;
            check = 0.0;

            // summation of x
            for (i = 0; i < n; i++)
                sumx += Double.Parse(dataLinear_xfx[1, i].Value.ToString());

            // summation of y
            for (i = 0; i < n; i++)
                sumy += Double.Parse(dataLinear_xfx[2, i].Value.ToString());

            // summation of xy
            for (i = 0; i < n; i++)
            {
                temp = Double.Parse(dataLinear_xfx[1, i].Value.ToString()) * Double.Parse(dataLinear_xfx[2, i].Value.ToString());
                sumxy += Math.Round(temp, 5);
            }
```

```
            // summation of x^2
            for (i = 0; i < n; i++)
            {
                temp = Double.Parse(dataLinear_xfx[1, i].Value.ToString()) * Double.Parse(dataLinear_xfx[1, i].Value.ToString());
                sumxx += Math.Round(temp, 5);
            }

            // solve for the coefficients
            a1 = (n * sumxy - sumx * sumy) / (n * sumxx - (sumx * sumx));
            a1 = Math.Round(a1, 5);

            a0 = (sumy / n) - a1 * (sumx / n);
            a0 = Math.Round(a0, 5);

            // check for zero division or NaN or inf
            if (double.IsNaN(a1))
            {
                dataLinear_xfxSize = 0;
                dataLinear_xfx.Rows.Clear();
                clearBox();

                MessageBox.Show("Hi. I'm sorry but we tried dividing by zero in solving for a1.\nIf you're still alive, then I want you to
try again, okay?", "ERR", MessageBoxButtons.OK);
                return;
            }

            // write the coefficients
            txtBoxLinear_a0.Text = a0.ToString();
            txtBoxLinear_a1.Text = a1.ToString();

            MessageBox.Show("Yay! I don't have anything to say but,\nIt's superbly awesome, yes?", "ANS", MessageBoxButtons.OK);
        }

        private void btnLinear_add_Click(object sender, EventArgs e)
        {
            // handle errors
            try
            {
                check = Double.Parse(txtBoxLinear_x.Text);
                check = Double.Parse(txtBoxLinear_fx.Text);
            }

            catch (FormatException)
            {
                clearBox();
                MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR",
MessageBoxButtons.OK);
                return;
            }

            // write data points
            dataLinear_xfx.Rows.Add(dataLinear_xfxSize, txtBoxLinear_x.Text.ToString(), txtBoxLinear_fx.Text.ToString());
            dataLinear_xfxSize++;

            // clear textboxes
            txtBoxLinear_x.Text = "X";
            txtBoxLinear_fx.Text = "f(X)";
            this.ActiveControl = txtBoxLinear_x;
```

```
        }

        private void btnLinear_solve_Click(object sender, EventArgs e)
        {
            // check if inputs > 0
            if (dataLinear_xfxSize > 0)
                // solve for the coefficients
                solve();
            else
            {
                clearBox();
                MessageBox.Show("Hi. I'm sorry but you lack inputs. It's okay.\nI want you to try again, okay?", "ERR",
MessageBoxButtons.OK);
                return;
            }
        }

        private void btnLinear_clear_Click(object sender, EventArgs e)
        {
            // clear everything
            txtBoxLinear_x.Text = "X";
            txtBoxLinear_fx.Text = "f(X)";

            clearBox();

            dataLinear_xfxSize = 0;
            dataLinear_xfx.Rows.Clear();
        }

        private void clearBox()
        {
            txtBoxLinear_a0.Text = "a0";
            txtBoxLinear_a1.Text = "a1";
        }
    }
}
```

APPENDIX

INTERPOLATION

NEWTON'S DIVIDED DIFFERENCE INTERPOLATING POLYNOMIALS

SOURCE CODE

FORM3.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;

namespace MP4
{
  public partial class frmNewton : Form
  {
    // globally initialize variables
    int dataNewton_xfxSize;
    double check;

    int n, test;
    static string temp;
    static decimal[,] List;
    decimal x0, x1, y0, y1;
    decimal tempVal;

    public frmNewton()
    {
      InitializeComponent();
    }

    public void GenerateFunction()
    {
      // reinitialize variables
      n = dataNewton_xfx.Rows.Count;

      // solve for the coefficients
      test = NewtonsInterpolation();

      if (test == 1)
      {
        return;
      }

      // store the equation in a string
      string fnc = dataNewton_xfx[2, 0].Value.ToString();

      for (int i = 0; i < (n - 1); i++)
      {
```

```
        if (Decimal.Parse(List[0, i].ToString()) > 0)
            fnc += " + ";
        else if (Decimal.Parse(List[0, i].ToString()) < 0)
            fnc += " - ";
        else
            continue;

        temp = "";

        for (int j = 0; j <= i; j++)
        {
            x0 = Decimal.Parse(dataNewton_xfx[1, j].Value.ToString());

            if (x0 > 0)
                temp += "(x - " + x0 + ")";
            else if (x0 < 0)
                temp += "(x + " + (x0 * -1) + ")";
            else
                temp += "(x)";
        }

        fnc += Math.Abs(List[0, i]).ToString();
        fnc += temp;
    }

    // write equation
    txtBoxNewton_eq.Text = fnc;

    MessageBox.Show("Yay! I don't have anything to say but,\nIt's superbly awesome, yes?", "ANS", MessageBoxButtons.OK);
}

public int NewtonsInterpolation()
{
    // create the table
    List = new decimal[n - 1, n - 1];

    for (int j = 0; j < n - 1; j++)
    {
        for (int i = 0; i < n - (j + 1); i++)
        {
            if (j == 0)
            {
                x0 = Decimal.Parse(dataNewton_xfx[1, i].Value.ToString());
                x1 = Decimal.Parse(dataNewton_xfx[1, i + 1].Value.ToString());
                y0 = Decimal.Parse(dataNewton_xfx[2, i].Value.ToString());
                y1 = Decimal.Parse(dataNewton_xfx[2, i + 1].Value.ToString());

                // check for zero division or NaN or inf
                try
                {
                    tempVal = (y1 - y0) / (x1 - x0);
                }

                catch (DivideByZeroException)
                {
                    txtBoxNewton_eq.Text = "EQUATION";

                    dataNewton_xfxSize = 0;
```

```
                    dataNewton_xfx.Rows.Clear();

                    MessageBox.Show("Hi. I'm sorry but we tried dividing by zero.\nIf you're still alive, then I want you to try again,
okay?", "ERR", MessageBoxButtons.OK);
                    return 1;
                }

                List[i, j] = Math.Round(tempVal, 5);
            } else
            {
                x0 = Decimal.Parse(dataNewton_xfx[1, i].Value.ToString());
                x1 = Decimal.Parse(dataNewton_xfx[1, i + (j + 1)].Value.ToString());

                // check for zero division or NaN or inf
                try
                {
                    tempVal = (List[(i + 1), (j - 1)] - List[i, j - +1]) / (x1 - x0);
                }

                catch (DivideByZeroException)
                {
                    txtBoxNewton_eq.Text = "EQUATION";

                    dataNewton_xfxSize = 0;
                    dataNewton_xfx.Rows.Clear();

                    MessageBox.Show("Hi. I'm sorry but we tried dividing by zero.\nIf you're still alive, then I want you to try again,
okay?", "ERR", MessageBoxButtons.OK);
                    return 1;
                }

                List[i, j] = Math.Round(tempVal, 5);
            }
        }
    }

    return 0;
}

private void btnNewton_add_Click(object sender, EventArgs e)
{
    // handle errors
    try
    {
        check = Double.Parse(txtBoxNewton_x.Text);
        check = Double.Parse(txtBoxNewton_fx.Text);
    }

    catch (FormatException)
    {
        txtBoxNewton_eq.Text = "EQUATION";
        MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR",
MessageBoxButtons.OK);
        return;
    }

    // write data points
    dataNewton_xfx.Rows.Add(dataNewton_xfxSize, txtBoxNewton_x.Text.ToString(), txtBoxNewton_fx.Text.ToString());
```

```
            dataNewton_xfxSize++;

            // clear textboxes
            txtBoxNewton_x.Text = "X";
            txtBoxNewton_fx.Text = "f(X)";

            this.ActiveControl = txtBoxNewton_x;
        }

        private void btnNewton_solve_Click(object sender, EventArgs e)
        {
            if (dataNewton_xfx.Rows.Count == 0)
            {
                txtBoxNewton_eq.Text = "EQUATION";
                MessageBox.Show("Hi. I'm sorry but you lack inputs. It's okay.\nI want you to try again, okay?", "ERR",
MessageBoxButtons.OK);
                return;
            }

            GenerateFunction();
        }

        private void btnNewton_clear_Click(object sender, EventArgs e)
        {
            // clear everything
            txtBoxNewton_x.Text = "X";
            txtBoxNewton_fx.Text = "f(X)";
            txtBoxNewton_eq.Text = "EQUATION";

            dataNewton_xfxSize = 0;
            dataNewton_xfx.Rows.Clear();
        }
    }
}
```

APPENDIX

NUMERICAL INTEGRATION

TRAPEZOIDAL RULE

SOURCE CODE

FORM4.CS

```
using System;
using System.Collections.Generic;
using System.Collections;
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;

namespace MP4
{
    public partial class frmTrapezoidal : Form
    {
        // globally initialize variables
        double[] xi;
        double[] fxi;
        double[] trapRule;
        double LLimit, ULimit;
        double h;
        double integral;
        int n;
        double sum;

        ExpressionParser myParse;
        Hashtable myHash;

        public frmTrapezoidal()
        {
            InitializeComponent();
        }

        public void SetData()
        {
            // write the intervals
            for (int i = 0; i < (n + 1); i++)
            {
                xi[i] = Math.Round(LLimit, 5);
                LLimit += h;
            }
        }

        public void useTrapRule()
        {
            // perform trapezoidal rule
```

```
            sum = 0;

            for (int i = 0; i < (n + 1); i++)
            {
                if (checkFirstLast(i))
                    trapRule[i] = Math.Round(fxi[i], 5);
                else if (!checkFirstLast(i))
                    trapRule[i] = Math.Round((2 * fxi[i]), 5);

                sum += trapRule[i];
            }

            integral = (h / 2) * sum;
            txtBoxTrapezoidal_i.Text = Math.Round(integral, 5).ToString("0.00000");

            // display result
            MessageBox.Show("Yay! The integral approximation of the function within the given limits is " + txtBoxTrapezoidal_i.Text
+ ".\nIt's superbly awesome, yes?", "ANS", MessageBoxButtons.OK);
        }

        public bool checkFirstLast(int i)
        {
            // check if fx is first or last
            if (i == 0 || i == n)
                return true;
            else
                return false;
        }

        private void btnTrapezoidal_solve_Click(object sender, EventArgs e)
        {
            // handle errors
            try
            {
                n = int.Parse(txtBoxTrapezoidal_n.Text);
                LLimit = Double.Parse(txtBoxTrapezoidal_a.Text);
                ULimit = Double.Parse(txtBoxTrapezoidal_b.Text);
            }
            catch (FormatException)
            {
                txtBoxTrapezoidal_i.Text = "I";
                MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR",
MessageBoxButtons.OK);
                return;
            }

            if(n <= 0)
            {
                txtBoxTrapezoidal_i.Text = "I";
                MessageBox.Show("Hi. I'm sorry but n cannot be zero or below. It's okay.\nI want you to try again, okay?", "ERR",
MessageBoxButtons.OK);
                return;
            }

            // reinitialize variables
            myParse = new ExpressionParser();
            myHash = new Hashtable();
```

```
            xi = new double[n + 1];
            fxi = new double[n + 1];
            trapRule = new double[n + 1];

            h = (ULimit - LLimit) / n;

            SetData();

            // equation parse
            for (int i = 0; i < (n + 1); i++)
            {
                // handle errors
                try
                {
                    myHash.Clear();
                    myHash.Add("x", xi[i].ToString());
                    fxi[i] = Math.Round(myParse.Parse(txtBoxTrapezoidal_eq.Text, myHash), 5);

                    // check for Nan or inf
                    if (double.IsNaN(fxi[i]) || double.IsInfinity(fxi[i]))
                    {
                        txtBoxTrapezoidal_i.Text = "I";
                        MessageBox.Show("Hi. I'm sorry but with the given function and limits,\nI can tell you that the result will be NaN
and I can't handle NaN.\nI want you to try again, okay?", "ERR", MessageBoxButtons.OK);
                        return;
                    }
                }

                catch
                {
                    txtBoxTrapezoidal_i.Text = "I";
                    MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR",
MessageBoxButtons.OK);
                    return;
                }
            }

            useTrapRule();
        }

        private void btnTrapezoidal_clear_Click(object sender, EventArgs e)
        {
            // clear everything
            txtBoxTrapezoidal_eq.Text = "EQUATION";
            txtBoxTrapezoidal_n.Text = "n";
            txtBoxTrapezoidal_a.Text = "a";
            txtBoxTrapezoidal_b.Text = "b";
            txtBoxTrapezoidal_i.Text = "I";
        }
    }
}
```

APPENDIX

NUMERICAL DIFFERENTIATION

CENTERED FINITE DIVIDED DIFFERENCE

SOURCE CODE

FORM5.CS

```
using System;
using System.Collections.Generic;
using System.Collections;
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;

namespace MP4
{
  public partial class frmCenter : Form
  {
    // globally initialize variables
    double xi, fxi;
    double xiadd, fxiadd;
    double ximinus, fximinus;
    double h;
    double derivative;

    ExpressionParser myParse;
    Hashtable myHash;

    public frmCenter()
    {
      InitializeComponent();
    }

    public void useCenter()
    {
      // solve for estimate
      derivative = (fxiadd - fximinus) / (xiadd - ximinus);
    }

    private void btnCenter_solve_Click(object sender, EventArgs e)
    {
      // handle errors
      try
      {
        h = Double.Parse(txtBoxCenter_h.Text);
        xi = Double.Parse(txtBoxCenter_xi.Text);
      }

      catch (FormatException)
```

```
        {
            txtBoxCenter_d.Text = "D";
            MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR",
MessageBoxButtons.OK);
            return;
        }

        myParse = new ExpressionParser();
        myHash = new Hashtable();

        // solve for xis
        xiadd = xi + h;
        ximinus = xi - h;

        // handle errors
        try
        {
            // solve for fxs
            myHash.Clear();
            myHash.Add("x", xi.ToString());
            fxi = myParse.Parse(txtBoxCenter_eq.Text, myHash);

            myHash.Clear();
            myHash.Add("x", xiadd.ToString());
            fxiadd = myParse.Parse(txtBoxCenter_eq.Text, myHash);

            myHash.Clear();
            myHash.Add("x", ximinus.ToString());
            fximinus = myParse.Parse(txtBoxCenter_eq.Text, myHash);

            // check for Nan or inf
            if (double.IsNaN(fxi) || double.IsInfinity(fxi) ||
                double.IsNaN(fxiadd) || double.IsInfinity(fxiadd) ||
                double.IsNaN(fximinus) || double.IsInfinity(fximinus))
            {
                txtBoxCenter_d.Text = "D";
                MessageBox.Show("Hi. I'm sorry but with the given function and value,\nI can tell you that the result will be NaN and
I can't handle NaN.\nI want you to try again, okay?", "ERR", MessageBoxButtons.OK);
                return;
            }
        }

        catch
        {
            txtBoxCenter_d.Text = "D";
            MessageBox.Show("Hi. I'm sorry but your input is invalid. It's okay.\nI want you to try again, okay?", "ERR",
MessageBoxButtons.OK);
            return;
        }

        useCenter();
        txtBoxCenter_d.Text = Math.Round(derivative, 5).ToString();

        // display result
        MessageBox.Show("Yay! The derivative approximation of the function at the given value is " + txtBoxCenter_d.Text +
".\nIt's superbly awesome, yes?", "ANS", MessageBoxButtons.OK);
    }
```

```
        private void btnCenter_clear_Click(object sender, EventArgs e)
        {
            // clear everything
            txtBoxCenter_eq.Text = "EQUATION";
            txtBoxCenter_xi.Text = "Xi";
            txtBoxCenter_h.Text = "h";
            txtBoxCenter_d.Text = "D";
        }
    }
}
```

APPENDIX

REFERENCES

lundin.info/mathparser

stackoverflow.com

msdn.microsoft.com

existing applications