SECANT METHOD

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import net.objecthunter.exp4j.*;
import net.objecthunter.exp4j.tokenizer.UnknownFunctionOrVariableException;
import org.jfree.chart.*;
import org.jfree.chart.annotations.XYTextAnnotation;
import org.jfree.chart.plot.*;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.xy.*;

public class SecantMethodSolver extends JFrame {
    private JTextField functionField, x0Field, x1Field, tolField, maxIterField;
    private JTable table;
    private DefaultTableModel tableModel;
    private ArrayList<Double> roots = new ArrayList<>();

    public SecantMethodSolver() {
        setTitle("Secant Method Solver");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(1300, 700);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout(10, 10));

        // Instructions Panel
        JTextArea instructions = new JTextArea(
    "📚 Secant Method Solver\n" +
    "Instructions:\n" +
    "- Enter f(x) using standard math syntax (e.g., x^3 - x - 2).\n" +
    "- Supported functions: sin, cos, tan, exp, log, abs, sqrt, etc.\n" +
    "- Use ^ for exponents (e.g., x^3 means x cubed).\n" +
    "- Provide two initial guesses x0 and x1, a tolerance, and max iterations.\n" +
    "- Click 'Compute' to find the roots.\n"
);
instructions.setEditable(false);
instructions.setBackground(new Color(240, 240, 240));
instructions.setBorder(BorderFactory.createTitledBorder("Instructions"));
add(instructions, BorderLayout.NORTH);

// Input Panel
JPanel inputPanel = new JPanel(new GridLayout(6, 2, 8, 8));
inputPanel.setBorder(BorderFactory.createTitledBorder("Input Parameters"));
functionField = createField(inputPanel, "Function f(x):", "x^3 - x - 2");
x0Field = createField(inputPanel, "Initial Guess x0:", "1.0");
x1Field = createField(inputPanel, "Initial Guess x1:", "2.0");
tolField = createField(inputPanel, "Tolerance:", "1e-5");
maxIterField = createField(inputPanel, "Max Iterations:", "20");

JButton computeButton = new JButton("Compute");
JButton helpButton = new JButton("Help");

inputPanel.add(computeButton);
inputPanel.add(helpButton);
add(inputPanel, BorderLayout.WEST);
```

```
    // Iteration Table
    String[] columns = {"Iteration", "x", "f(x)", "Error"};
    tableModel = new DefaultTableModel(columns, 0);
    table = new JTable(tableModel);
    JScrollPane tableScroll = new JScrollPane(table);
    tableScroll.setBorder(BorderFactory.createTitledBorder("Iteration Table"));
    add(tableScroll, BorderLayout.CENTER);

    // Actions
    computeButton.addActionListener(e -> runSecantMethod());
    helpButton.addActionListener(e -> showHelpDialog());

    setVisible(true);
  }

  private JTextField createField(JPanel panel, String label, String defaultText) {
    JLabel lbl = new JLabel(label);
    JTextField field = new JTextField(defaultText);
    panel.add(lbl);
    panel.add(field);
    return field;
  }

  private void runSecantMethod() {
    tableModel.setRowCount(0);
    roots.clear();
    try {
      String fxExpr = functionField.getText().trim();
      // Warn user if they accidentally use Math.pow
      if (fxExpr.contains("Math.pow")) {
        JOptionPane.showMessageDialog(this,
            "⚠ Use ^ for exponents, not Math.pow!\nExample: x^3 - x - 2",
            "Syntax Warning", JOptionPane.WARNING_MESSAGE);
        return;
      }

      // Replace ^ with ** for exp4j (it supports ^ for exponents)
      Expression fx = new ExpressionBuilder(fxExpr)
                .variable("x")
                .build();

      double x0 = Double.parseDouble(x0Field.getText().trim());
      double x1 = Double.parseDouble(x1Field.getText().trim());
      double tol = Double.parseDouble(tolField.getText().trim());
      int maxIter = Integer.parseInt(maxIterField.getText().trim());

      double error = Double.MAX_VALUE;
      int iter = 1;

      while (iter <= maxIter && error > tol) {
        double fx0 = fx.setVariable("x", x0).evaluate();
        double fx1 = fx.setVariable("x", x1).evaluate();

        if (Math.abs(fx1 - fx0) < 1e-12) {
          JOptionPane.showMessageDialog(this, "Error: Division by zero in Secant formula.", "Error",
JOptionPane.ERROR_MESSAGE);
          return;
        }
```

```
                double x2 = x1 - fx1 * (x1 - x0) / (fx1 - fx0);
                double fx2 = fx.setVariable("x", x2).evaluate();
                error = Math.abs(x2 - x1);

                tableModel.addRow(new Object[] {
                    iter, String.format("%.8f", x2), String.format("%.8f", fx2), String.format("%.8f", error)
                });

                x0 = x1;
                x1 = x2;
                iter++;
            }

            roots.add(x1);
            JOptionPane.showMessageDialog(this, String.format("[OK] Root found at x ≈ %.8f", x1), "Result", JOptionPane.INFORMATION_MESSAGE);

            showPlot(fx, x1);

        } catch (UnknownFunctionOrVariableException ufve) {
            JOptionPane.showMessageDialog(this, "Error: Unknown function or variable in your expression.\n" +
                    "Use functions like sin(x), cos(x), exp(x), log(x), etc.", "Error", JOptionPane.ERROR_MESSAGE);
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        }
    }
```

```java
private void showPlot(Expression fx, double xApprox) {
    XYSeries seriesF = new XYSeries("f(x)");

    double xMin = xApprox - 5, xMax = xApprox + 5;
    for (double x = xMin; x <= xMax; x += 0.01) {
        seriesF.add(x, fx.setVariable("x", x).evaluate());
    }

    XYSeriesCollection dataset = new XYSeriesCollection(seriesF);
    JFreeChart chart = ChartFactory.createXYLineChart(
        "Secant Method: Root Finding",
        "x",
        "f(x)",
        dataset,
        PlotOrientation.VERTICAL,
        true,
        true,
        false
    );

    XYPlot plot = chart.getXYPlot();
    plot.setDomainGridlinesVisible(true);
    plot.setRangeGridlinesVisible(true);

    XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer(true, false);
    renderer.setSeriesPaint(0, Color.BLUE);
    plot.setRenderer(renderer);

    for (double root : roots) {
      XYSeries rootPoint = new XYSeries("Root");
      rootPoint.add(root, 0);
      XYSeriesCollection rootDataset = new XYSeriesCollection(rootPoint);

      XYLineAndShapeRenderer rootRenderer = new XYLineAndShapeRenderer(false, true);
      rootRenderer.setSeriesPaint(0, Color.RED);
      rootRenderer.setSeriesShape(0, new java.awt.geom.Ellipse2D.Double(-5, -5, 10, 10));

      int datasetIndex = plot.getDatasetCount();
      plot.setDataset(datasetIndex, rootDataset);
      plot.setRenderer(datasetIndex, rootRenderer);

      XYTextAnnotation annotation = new XYTextAnnotation(String.format("%.5f", root), root, 0.05);
      annotation.setFont(new Font("SansSerif", Font.BOLD, 12));
      annotation.setPaint(Color.RED);
      plot.addAnnotation(annotation);
    }

    SwingUtilities.invokeLater(() -> {
      JFrame plotFrame = new JFrame("Secant Method Plot");
      plotFrame.setSize(900, 600);
      plotFrame.add(new ChartPanel(chart));
      plotFrame.setLocationRelativeTo(null);
      plotFrame.setVisible(true);
    });
  }

  private void showHelpDialog() {
    String message = """
```

```
            💥 Secant Method Overview 💥

            - The Secant Method is an iterative technique for finding roots of nonlinear equations.
            - Formula: x_{n+1} = x_n - f(x_n) * (x_n - x_{n-1}) / (f(x_n) - f(x_{n-1}))
            - Provide two initial guesses (x0, x1), tolerance, and max iterations.

            📝 Function Syntax:
            - Use standard math functions: x^3 - x - 2
            - Supported functions: sin(x), cos(x), tan(x), exp(x), log(x), abs(x), sqrt(x), etc.
            - Use ^ for exponents (e.g., x^3 for x cubed).

            📌 Example:
            Function: x^3 - x - 2
            x0: 1.0
            x1: 2.0
            Tolerance: 1e-5
            Max Iterations: 20
            """;
        JOptionPane.showMessageDialog(this, message, "Secant Method Help", JOptionPane.INFORMATION_MESSAGE);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(SecantMethodSolver::new);
    }
}
```

GRAPHICAL METHOD

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import java.util.List;
import net.objecthunter.exp4j.Expression;
import net.objecthunter.exp4j.ExpressionBuilder;
import org.jfree.chart.*;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.annotations.XYTextAnnotation;
import org.jfree.chart.renderer.xy.XYSplineRenderer;
import org.jfree.data.xy.*;

public class GraphicalMethodApp extends JFrame {
  private JTextField funcField, iterField;
  private JTable table;

  public GraphicalMethodApp() {
    setTitle("Graphical Method Root Finder");
    setSize(900, 700);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setLocationRelativeTo(null);
    setLayout(new BorderLayout());

    // === Input Panel ===
    JPanel inputPanel = new JPanel(new GridLayout(3, 2, 10, 10));
    inputPanel.setBorder(BorderFactory.createTitledBorder("Enter Inputs"));
    funcField = new JTextField("");
    iterField = new JTextField("");

    inputPanel.add(new JLabel("Enter function f(x):"));
    inputPanel.add(funcField);
    inputPanel.add(new JLabel("Max Iterations:"));
    inputPanel.add(iterField);

    JButton enterButton = new JButton("ENTER");
    JButton aboutButton = new JButton("ABOUT");
    inputPanel.add(enterButton);
    inputPanel.add(aboutButton);

    // === Table Panel ===
    table = new JTable();
    JScrollPane tableScroll = new JScrollPane(table);
    tableScroll.setBorder(BorderFactory.createTitledBorder("Result Table"));
    tableScroll.setPreferredSize(new Dimension(850, 300));

    add(inputPanel, BorderLayout.NORTH);
    add(tableScroll, BorderLayout.CENTER);

    // === Actions ===
    enterButton.addActionListener(e -> process());
    aboutButton.addActionListener(e -> showAboutDialog());

    setVisible(true);
  }
```

```
    private void process() {
        try {
            String input = funcField.getText().trim();
            input = input.replaceAll("@\\(x\\)", "");  // Remove @(x)
            input = input.replaceAll("\\.\\^", "\\^");  // Replace .^ with ^
            input = input.replaceAll("\\.\\*", "*");    // Replace .* with *
            input = input.replaceAll("\\./", "/");      // Replace ./ with /

            int maxIter = Integer.parseInt(iterField.getText().trim());
            if (maxIter <= 0) {
                JOptionPane.showMessageDialog(this, "Max iterations must be positive.", "Input Error",
JOptionPane.ERROR_MESSAGE);
                return;
            }

            double a = -10;  // Fixed interval for simplicity
            double b = 10;
            double h = (b - a) / maxIter;

            Expression exp = new ExpressionBuilder(input).variable("x").build();
            List<Double> xVals = new ArrayList<>();
            List<Double> yVals = new ArrayList<>();
            List<Double> roots = new ArrayList<>();

            for (int i = 0; i <= maxIter; i++) {
                double x = a + i * h;
                double y = exp.setVariable("x", x).evaluate();
                xVals.add(x);
                yVals.add(y);
            }

            double tol = 1e-5;
            for (int i = 1; i < yVals.size(); i++) {
                if (yVals.get(i - 1) * yVals.get(i) <= 0) {
                    double root = refineRoot(exp, xVals.get(i - 1), xVals.get(i), tol);
                    if (roots.stream().noneMatch(r -> Math.abs(r - root) < tol)) {
                        roots.add(root);
                    }
                }
            }

            // Populate table
            String[][] data = new String[xVals.size()][2];
            for (int i = 0; i < xVals.size(); i++) {
                data[i][0] = String.format("%.5f", xVals.get(i));
                data[i][1] = String.format("%.7f", yVals.get(i));
            }
            table.setModel(new javax.swing.table.DefaultTableModel(data, new String[]{"x", "f(x)"}));

            // Plot the function
            plotGraph(input, xVals, yVals, roots);

        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(this, "Please enter a valid number for max iterations.", "Input Error", JOptionPane.ERROR_MESSAGE);
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage(), "Input Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    private double refineRoot(Expression exp, double x1, double x2, double tol) {
        double mid;
        while ((x2 - x1) > tol) {
            mid = (x1 + x2) / 2;
            double f1 = exp.setVariable("x", x1).evaluate();
            double fmid = exp.setVariable("x", mid).evaluate();
            if (f1 * fmid <= 0) x2 = mid;
            else x1 = mid;
        }
        return (x1 + x2) / 2;
    }

    private void plotGraph(String func, List<Double> xVals, List<Double> yVals, List<Double> roots) {
        XYSeries series = new XYSeries("f(x)");
        for (int i = 0; i < xVals.size(); i++) {
            series.add(xVals.get(i), yVals.get(i));
        }

        XYSeriesCollection dataset = new XYSeriesCollection(series);
        JFreeChart chart = ChartFactory.createXYLineChart("Graphical Method", "x", "f(x)", dataset);
        XYPlot plot = chart.getXYPlot();
        plot.setRenderer(new XYSplineRenderer());

        // Add X-axis marker at y=0
        plot.addRangeMarker(new org.jfree.chart.plot.ValueMarker(0.0, Color.BLACK, new
BasicStroke(1.5f)));
```

```
// Mark roots with red dots
XYSeries rootSeries = new XYSeries("Roots");
for (double root : roots) {
    rootSeries.add(root, 0.0);
}
dataset.addSeries(rootSeries);

XYSplineRenderer renderer = new XYSplineRenderer();
renderer.setSeriesPaint(0, Color.BLUE); // Function line
renderer.setSeriesPaint(1, Color.RED);  // Root dots
renderer.setSeriesShapesVisible(1, true);
renderer.setSeriesLinesVisible(1, false);
renderer.setSeriesShape(1, new java.awt.geom.Ellipse2D.Double(-4, -4, 8, 8));
plot.setRenderer(renderer);

// Text annotations for roots
for (int i = 0; i < roots.size(); i++) {
    double root = roots.get(i);
    XYTextAnnotation ann = new XYTextAnnotation("Root " + (i + 1), root, 0);
    ann.setPaint(Color.RED);
    ann.setFont(new Font("Arial", Font.BOLD, 12));
    plot.addAnnotation(ann);
}

// Display graph
JFrame frame = new JFrame("Graphical Method");
frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
frame.setSize(850, 650);
```

```
        frame.add(new ChartPanel(chart));
        frame.setVisible(true);
    }

    private void showAboutDialog() {
        String message = """
                === Graphical Method Root Finder ===

                Instructions:
                1. Enter the function using JavaScript-like syntax.
                   Example: 2*x^2 - 5*x + 3
                2. Enter the maximum number of iterations.
                   (More iterations give better graph resolution)
                3. The program scans the interval [-10, 10].
                4. The function is plotted, and roots (where sign changes) are marked in red.

                Note: Use element-wise operators:
                      ^ for power, * for multiplication, / for division.

                """;
        JOptionPane.showMessageDialog(this, message, "About - Graphical Method", JOptionPane.INFORMATION_MESSAGE);
    }

    public static void main(String[] args) {
        try {
            for (UIManager.LookAndFeelInfo info : UIManager.getInstalledLookAndFeels())
                if ("Nimbus".equals(info.getName())) UIManager.setLookAndFeel(info.getClassName());
        } catch (Exception ignored) {}
        SwingUtilities.invokeLater(GraphicalMethodApp::new);
    }
}
```

BISECTION METHOD

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import net.objecthunter.exp4j.*;
import org.jfree.chart.*;
import org.jfree.chart.plot.*;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.xy.*;

public class BisectionMethodSolver extends JFrame {
  private JTextField functionField, aField, bField, tolField, iterField;
  private JTable iterationTable;
  private DefaultTableModel tableModel;
  private ArrayList<Double> roots = new ArrayList<>();
  private String equation;

  public BisectionMethodSolver() {
    setTitle("Bisection Method Solver - Enhanced Version");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(1200, 700);
    setLocationRelativeTo(null);
    setLayout(new BorderLayout(10, 10));

    // 🔷 Instructions Panel
    JTextArea instructions = new JTextArea(
      "📚 Bisection Method Solver\n" +
    "Instructions:\n" +
    "1. Enter a valid equation in terms of x (e.g., x^3 - x - 2).\n" +
    "2. Enter an interval [a, b] such that f(a)*f(b) < 0.\n" +
    "3. Enter the tolerance (e.g., 1e-5) and max iterations.\n" +
    "4. Click 'Compute' to find the root, view the table, and plot.\n"
);
instructions.setEditable(false);
instructions.setBackground(new Color(240, 240, 240));
instructions.setBorder(BorderFactory.createTitledBorder("How to Use"));
add(instructions, BorderLayout.NORTH);

// ◈ Input Panel
JPanel inputPanel = new JPanel(new GridLayout(6, 2, 8, 8));
inputPanel.setBorder(BorderFactory.createTitledBorder("Input Parameters"));
functionField = createField(inputPanel, "Function f(x):", "x^3 - x - 2");
aField = createField(inputPanel, "Interval Start (a):", "1");
bField = createField(inputPanel, "Interval End (b):", "2");
tolField = createField(inputPanel, "Tolerance:", "1e-5");
iterField = createField(inputPanel, "Max Iterations:", "100");

JButton computeButton = new JButton("Compute");
inputPanel.add(computeButton);
add(inputPanel, BorderLayout.WEST);

// ◈ Table for Iteration Log
String[] columnNames = {"Iter", "xl", "xr", "xu", "f(xl)", "f(xr)", "Error", "f(xl)*f(xr)", "Remarks"};
tableModel = new DefaultTableModel(columnNames, 0);
iterationTable = new JTable(tableModel);
JScrollPane tableScroll = new JScrollPane(iterationTable);
```

```
        tableScroll.setBorder(BorderFactory.createTitledBorder("Bisection Iteration Table"));
        add(tableScroll, BorderLayout.CENTER);

        // ◈ Compute Button Action
        computeButton.addActionListener(e -> runBisection());

        setVisible(true);
    }

    private JTextField createField(JPanel panel, String label, String defaultText) {
        JLabel lbl = new JLabel(label);
        JTextField field = new JTextField(defaultText);
        panel.add(lbl);
        panel.add(field);
        return field;
    }

    private double evaluate(double x) throws Exception {
        Expression expr = new ExpressionBuilder(equation).variable("x").build().setVariable("x", x);
        return expr.evaluate();
    }

    private void runBisection() {
        tableModel.setRowCount(0);
        roots.clear();
        try {
            equation = functionField.getText().trim();
            double xl = Double.parseDouble(aField.getText().trim());
            double xr = Double.parseDouble(bField.getText().trim());
        ;

        double tol = Double.parseDouble(tolField.getText().trim());
        int maxIter = Integer.parseInt(iterField.getText().trim());

        double fxl = evaluate(xl);
        double fxr = evaluate(xr);
        if (fxl * fxr >= 0) {
          JOptionPane.showMessageDialog(this, "Error: f(a) * f(b) must be negative.", "Error", JOptionPane.ERROR_MESSAGE);
          return;
        }

        double xu = 0, fxu, error;
        for (int iter = 1; iter <= maxIter; iter++) {
          xu = (xl + xr) / 2;
          fxu = evaluate(xu);
          error = Math.abs(xr - xl) / 2;
          double product = fxl * fxu;

          String remarks = "";
          if (Math.abs(fxu) < tol || error < tol) {
            remarks = "Converged ✅";
          } else {
            remarks = (product < 0) ? "Root in [xl,xu]" : "Root in [xu,xr]";
          }

          roots.add(xu);
          tableModel.addRow(new Object[]{
            iter, xl, xr, xu, fxl, fxr, error, product, remarks
          });
```

```
                if (Math.abs(fxu) < tol || error < tol) break;

                if (product < 0) {
                    xr = xu;
                    fxr = fxu;
                } else {
                    xl = xu;
                    fxl = fxu;
                }
            }

            showPlot(Double.parseDouble(aField.getText().trim()), 
Double.parseDouble(bField.getText().trim()));
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage(), "Error", 
JOptionPane.ERROR_MESSAGE);
        }
    }

    private void showPlot(double a, double b) {
    XYSeries seriesF = new XYSeries("f(x)");
    XYSeries seriesRoots = new XYSeries("Root Approximations");

    for (double x = a - 0.1 * Math.abs(b - a); x <= b + 0.1 * Math.abs(b - a); x += (b - a) / 200.0) {
        try {
            seriesF.add(x, evaluate(x));
        } catch (Exception ignored) {}
    }
```

```
for (double root : roots) {
    seriesRoots.add(root, 0);
}

XYSeriesCollection dataset = new XYSeriesCollection();
dataset.addSeries(seriesF);
dataset.addSeries(seriesRoots);

JFreeChart chart = ChartFactory.createXYLineChart(
    "Bisection Method - Root Finding",
    "x",
    "f(x)",
    dataset,
    PlotOrientation.VERTICAL,
    true,
    true,
    false
);

XYPlot plot = chart.getXYPlot();
plot.setDomainGridlinesVisible(true);
plot.setRangeGridlinesVisible(true);

XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer();
renderer.setSeriesLinesVisible(0, true);  // f(x) line
renderer.setSeriesShapesVisible(0, false);
renderer.setSeriesLinesVisible(1, false); // Dots for roots only
renderer.setSeriesShapesVisible(1, true);
```

```
    renderer.setSeriesShape(1, new java.awt.geom.Ellipse2D.Double(-4, -4, 8, 8));
    renderer.setSeriesPaint(0, Color.BLUE);
    renderer.setSeriesPaint(1, Color.RED);

    plot.setRenderer(renderer);

    SwingUtilities.invokeLater(() -> {
        JFrame plotFrame = new JFrame("Bisection Plot");
        plotFrame.setSize(800, 600);
        plotFrame.add(new ChartPanel(chart));
        plotFrame.setLocationRelativeTo(null);
        plotFrame.setVisible(true);
    });
}

    public static void main(String[] args) {
        SwingUtilities.invokeLater(BisectionMethodSolver::new);
    }
}
```

INCREMENTAL METHOD

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.util.ArrayList;
import net.objecthunter.exp4j.*;
import org.jfree.chart.*;
import org.jfree.chart.annotations.XYTextAnnotation;
import org.jfree.chart.plot.*;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.xy.*;

public class IncrementalMethodSolver extends JFrame {
  private JTextField functionField, aField, bField, hField;
  private JTable table;
  private DefaultTableModel tableModel;
  private ArrayList<Double> roots = new ArrayList<>();
  private String equation;

  public IncrementalMethodSolver() {
    setTitle("Incremental Method Root Finder");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(1300, 700);
    setLocationRelativeTo(null);
    setLayout(new BorderLayout(10, 10));

    // Instructions Panel
    JTextArea instructions = new JTextArea(
      "📚 Incremental Method Solver\n" +
    "Instructions:\n" +
    "1. Enter a valid equation in terms of x (e.g., x^3 - x - 2).\n" +
    "2. Enter an interval [a, b] such that a < b.\n" +
    "3. Enter a small positive step size h (e.g., 0.1).\n" +
    "4. Click 'Compute' to estimate roots.\n"
);
instructions.setEditable(false);
instructions.setBackground(new Color(240, 240, 240));
instructions.setBorder(BorderFactory.createTitledBorder("How to Use"));
add(instructions, BorderLayout.NORTH);

// Input Panel
JPanel inputPanel = new JPanel(new GridLayout(5, 2, 8, 8));
inputPanel.setBorder(BorderFactory.createTitledBorder("Input Parameters"));
functionField = createField(inputPanel, "Function f(x):", "x^3 - x - 2");
aField = createField(inputPanel, "Interval Start (a):", "0");
bField = createField(inputPanel, "Interval End (b):", "5");
hField = createField(inputPanel, "Increment h:", "0.1");

JButton computeButton = new JButton("Compute");
JButton helpButton = new JButton("Help");

inputPanel.add(computeButton);
inputPanel.add(helpButton);

add(inputPanel, BorderLayout.WEST);

// Table for Iteration Log (new columns)
String[] columns = {"Iteration", "xl", "deltax", "xu", "f(xl)", "f(xu)*f(xl)", "Remark"};
```

```
    tableModel = new DefaultTableModel(columns, 0);
    table = new JTable(tableModel);
    JScrollPane tableScroll = new JScrollPane(table);
    tableScroll.setBorder(BorderFactory.createTitledBorder("Iteration Table"));
    add(tableScroll, BorderLayout.CENTER);

    computeButton.addActionListener(e -> runIncremental());

    helpButton.addActionListener(e -> showHelpDialog());

    setVisible(true);
}

private JTextField createField(JPanel panel, String label, String defaultText) {
    JLabel lbl = new JLabel(label);
    JTextField field = new JTextField(defaultText);
    panel.add(lbl);
    panel.add(field);
    return field;
}

private double evaluate(double x) throws Exception {
    Expression expr = new ExpressionBuilder(equation).variable("x").build().setVariable("x", x);
    return expr.evaluate();
}

private void runIncremental() {
    tableModel.setRowCount(0);
    roots.clear();
```

```java
        try {
            equation = functionField.getText().trim();
            double a = Double.parseDouble(aField.getText().trim());
            double b = Double.parseDouble(bField.getText().trim());
            double h = Double.parseDouble(hField.getText().trim());

            if (h <= 0 || a >= b || h >= (b - a)) {
                JOptionPane.showMessageDialog(this, "[!] Invalid input. Ensure:\n- a < b\n- h > 0\n- h < (b - a).", "Input Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            ArrayList<Double> xValues = new ArrayList<>();
            ArrayList<Double> fxValues = new ArrayList<>();

            // Fill values & table rows for intervals
            int iteration = 1;
            for (double x = a; x <= b; x += h) {
                double fx = evaluate(x);
                xValues.add(x);
                fxValues.add(fx);
            }

            for (int i = 0; i < xValues.size() - 1; i++) {
                double xl = xValues.get(i);
                double xu = xValues.get(i + 1);
                double fx_l = fxValues.get(i);
                double fx_u = fxValues.get(i + 1);
                double prod = fx_l * fx_u;
                String remark = prod <= 0 ? "Root Interval" : "-";

                if (prod <= 0) {
                    double approxRoot = (xl + xu) / 2;
                    roots.add(approxRoot);
                }

                tableModel.addRow(new Object[]{
                    iteration++,
                    String.format("%.6f", xl),
                    String.format("%.6f", h),
                    String.format("%.6f", xu),
                    String.format("%.6f", fx_l),
                    String.format("%.6f", prod),
                    remark
                });
            }

            if (roots.isEmpty()) {
                JOptionPane.showMessageDialog(this, "[!] No root found in the interval.", "Result",
    JOptionPane.INFORMATION_MESSAGE);
            } else {
                StringBuilder result = new StringBuilder("[OK] Estimated roots at:\n");
                for (double root : roots) {
                    result.append(String.format("Root at x ≈ %.6f\n", root));
                }
                JOptionPane.showMessageDialog(this, result.toString(), "Result",
    JOptionPane.INFORMATION_MESSAGE);
            }
```

```
            showPlot(a, b, xValues, fxValues);
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        }
    }

    private void showPlot(double a, double b, ArrayList<Double> xVals, ArrayList<Double> fxVals) {
        // Function series (line only)
        XYSeries seriesF = new XYSeries("f(x)");
        for (int i = 0; i < xVals.size(); i++) {
            seriesF.add(xVals.get(i), fxVals.get(i));
        }

        XYSeriesCollection dataset = new XYSeriesCollection();
        dataset.addSeries(seriesF);

        JFreeChart chart = ChartFactory.createXYLineChart(
            "Incremental Method - Root Finding",
            "x",
            "f(x)",
            dataset,
            PlotOrientation.VERTICAL,
            true,
            true,
            false
        );
```

```
XYPlot plot = chart.getXYPlot();
plot.setDomainGridlinesVisible(true);
plot.setRangeGridlinesVisible(true);

// Renderer for function: line only, blue
XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer(true, false);
renderer.setSeriesPaint(0, Color.BLUE);
plot.setRenderer(renderer);

// Plot roots as red dots with labels near dots
for (double root : roots) {
    XYSeries rootPoint = new XYSeries("Root");
    rootPoint.add(root, 0);
    XYSeriesCollection rootDataset = new XYSeriesCollection(rootPoint);

    XYLineAndShapeRenderer rootRenderer = new XYLineAndShapeRenderer(false, true);
    rootRenderer.setSeriesPaint(0, Color.RED);
    rootRenderer.setSeriesShape(0, new java.awt.geom.Ellipse2D.Double(-5, -5, 10, 10));

    int datasetIndex = plot.getDatasetCount();
    plot.setDataset(datasetIndex, rootDataset);
    plot.setRenderer(datasetIndex, rootRenderer);

    // Add label near the dot
    XYTextAnnotation annotation = new XYTextAnnotation(
        String.format("%.4f", root),
        root,
        0.02 * (plot.getRangeAxis().getUpperBound() - plot.getRangeAxis().getLowerBound())
    );
```

```
            annotation.setFont(new Font("SansSerif", Font.BOLD, 12));
            annotation.setPaint(Color.RED);
            plot.addAnnotation(annotation);
        }

        SwingUtilities.invokeLater(() -> {
            JFrame plotFrame = new JFrame("Incremental Plot");
            plotFrame.setSize(900, 600);
            plotFrame.add(new ChartPanel(chart));
            plotFrame.setLocationRelativeTo(null);
            plotFrame.setVisible(true);
        });
    }

    private void showHelpDialog() {
        String helpMessage = """
            📚 Incremental Method Root Finder Help

            Instructions:
            1. Enter a valid equation in terms of x (e.g., x^3 - x - 2).
            2. Enter an interval [a, b] such that a < b.
            3. Enter a small positive step size h (e.g., 0.1).
            4. Click 'Compute' to estimate roots.

            What is the Incremental Method?
            -------------------------------
            The Incremental Method is a root-finding technique that evaluates the function at
            equally spaced points in the given interval [a, b]. It detects roots by checking
            where the function changes sign between these points.
            Uses:
            - Simple way to locate approximate root intervals.
            - Good for functions where derivatives are hard to compute.
            - Helps provide initial guesses for other root-finding methods.

            Note: Smaller step sizes (h) yield better root approximations but require
            more computations.
            """;

        JOptionPane.showMessageDialog(this, helpMessage, "Help - Incremental Method", JOptionPane.INFORMATION_MESSAGE);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(IncrementalMethodSolver::new);
    }
}
```

NEWTON RAPHSON

```
import javax.swing.*;

import javax.swing.table.DefaultTableModel;

import java.awt.*;

import java.awt.event.*;

import java.util.*;

import net.objecthunter.exp4j.*;

import org.jfree.chart.*;

import org.jfree.chart.annotations.XYTextAnnotation;

import org.jfree.chart.plot.*;

import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;

import org.jfree.data.xy.*;


public class NewtonRaphsonSolver extends JFrame {

    private JTextField functionField, derivativeField, x0Field, tolField, maxIterField;

    private JTable table;

    private DefaultTableModel tableModel;

    private ArrayList<Double> roots = new ArrayList<>();


    public NewtonRaphsonSolver() {

        setTitle("Newton-Raphson Method Solver");

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setSize(1300, 700);

        setLocationRelativeTo(null);

        setLayout(new BorderLayout(10, 10));


        // Instructions Panel

        JTextArea instructions = new JTextArea(

            "📚 Newton-Raphson Method Solver\n" +
    "Instructions:\n" +
    "- Enter f(x) and f'(x) using math-like syntax (e.g., x^3 - x - 2).\n" +
    "- Supported functions: sin(x), cos(x), tan(x), log(x), ln(x), exp(x), abs(x), etc.\n" +
    "- For powers, use ^ (e.g., x^3 for x cubed).\n" +
    "- Provide an initial guess x0, a tolerance (e.g., 1e-5), and max iterations.\n" +
    "- Click 'Compute' to find the roots.\n"
);
instructions.setEditable(false);
instructions.setBackground(new Color(240, 240, 240));
instructions.setBorder(BorderFactory.createTitledBorder("Instructions"));
add(instructions, BorderLayout.NORTH);

// Input Panel
JPanel inputPanel = new JPanel(new GridLayout(7, 2, 8, 8));
inputPanel.setBorder(BorderFactory.createTitledBorder("Input Parameters"));
functionField = createField(inputPanel, "Function f(x):", "x^3 - x - 2");
derivativeField = createField(inputPanel, "Derivative f'(x):", "3*x^2 - 1");
x0Field = createField(inputPanel, "Initial Guess (x0):", "1.5");
tolField = createField(inputPanel, "Tolerance:", "1e-5");
maxIterField = createField(inputPanel, "Max Iterations:", "20");

JButton computeButton = new JButton("Compute");
JButton helpButton = new JButton("Help");

inputPanel.add(computeButton);
inputPanel.add(helpButton);
add(inputPanel, BorderLayout.WEST);

// Iteration Table
```

```
    String[] columns = {"Iteration", "x", "f(x)", "Error"};
    tableModel = new DefaultTableModel(columns, 0);
    table = new JTable(tableModel);
    JScrollPane tableScroll = new JScrollPane(table);
    tableScroll.setBorder(BorderFactory.createTitledBorder("Iteration Table"));
    add(tableScroll, BorderLayout.CENTER);

    computeButton.addActionListener(e -> runNewtonRaphson());

    helpButton.addActionListener(e -> showHelp());

    setVisible(true);
}

private JTextField createField(JPanel panel, String label, String defaultText) {
    JLabel lbl = new JLabel(label);
    JTextField field = new JTextField(defaultText);
    panel.add(lbl);
    panel.add(field);
    return field;
}

private void runNewtonRaphson() {
    tableModel.setRowCount(0);
    roots.clear();
    try {
        String fxExpr = functionField.getText().trim();
        String dfxExpr = derivativeField.getText().trim();
        double x0 = Double.parseDouble(x0Field.getText().trim());
        double tol = Double.parseDouble(tolField.getText().trim());
        int maxIter = Integer.parseInt(maxIterField.getText().trim());

        Expression fx = new ExpressionBuilder(fxExpr).variable("x").build();
        Expression dfx = new ExpressionBuilder(dfxExpr).variable("x").build();

        double x = x0, error = Double.MAX_VALUE;
        int iter = 1;

        while (iter <= maxIter && error > tol) {
            double fxVal = fx.setVariable("x", x).evaluate();
            double dfxVal = dfx.setVariable("x", x).evaluate();

            if (Math.abs(dfxVal) < 1e-12) {
                JOptionPane.showMessageDialog(this, "Derivative too close to zero at x = " + x, "Error",
JOptionPane.ERROR_MESSAGE);
                return;
            }

            double xNext = x - fxVal / dfxVal;
            error = Math.abs(xNext - x);

            tableModel.addRow(new Object[]{
                iter, String.format("%.8f", x), String.format("%.8f", fxVal), String.format("%.8f", error)
            });

            x = xNext;
            iter++;
        }
```

```
      roots.add(x);
      JOptionPane.showMessageDialog(this, String.format("[OK] Root found at x ≈ %.8f", x), "Result",
JOptionPane.INFORMATION_MESSAGE);

      showPlot(fx, x0);

    } catch (Exception ex) {
      JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
    }
  }

  private void showPlot(Expression fx, double x0) {
    XYSeries seriesF = new XYSeries("f(x)");

    double xMin = x0 - 5, xMax = x0 + 5;
    for (double x = xMin; x <= xMax; x += 0.01) {
      seriesF.add(x, fx.setVariable("x", x).evaluate());
    }

    XYSeriesCollection dataset = new XYSeriesCollection(seriesF);
    JFreeChart chart = ChartFactory.createXYLineChart(
      "Newton-Raphson Method",
      "x",
      "f(x)",
      dataset,
      PlotOrientation.VERTICAL,
      true,
    true,
    false
);

XYPlot plot = chart.getXYPlot();
plot.setDomainGridlinesVisible(true);
plot.setRangeGridlinesVisible(true);

XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer(true, false);
renderer.setSeriesPaint(0, Color.BLUE);
plot.setRenderer(renderer);

// Add root points and labels
for (double root : roots) {
    XYSeries rootPoint = new XYSeries("Root");
    rootPoint.add(root, 0);
    XYSeriesCollection rootDataset = new XYSeriesCollection(rootPoint);

    XYLineAndShapeRenderer rootRenderer = new XYLineAndShapeRenderer(false, true);
    rootRenderer.setSeriesPaint(0, Color.RED);
    rootRenderer.setSeriesShape(0, new java.awt.geom.Ellipse2D.Double(-5, -5, 10, 10));

    int datasetIndex = plot.getDatasetCount();
    plot.setDataset(datasetIndex, rootDataset);
    plot.setRenderer(datasetIndex, rootRenderer);

    XYTextAnnotation annotation = new XYTextAnnotation(String.format("%.5f", root), root, 0.05);
    annotation.setFont(new Font("SansSerif", Font.BOLD, 12));
    annotation.setPaint(Color.RED);
```

```
            plot.addAnnotation(annotation);
        }

        SwingUtilities.invokeLater(() -> {
            JFrame plotFrame = new JFrame("Newton-Raphson Plot");
            plotFrame.setSize(900, 600);
            plotFrame.add(new ChartPanel(chart));
            plotFrame.setLocationRelativeTo(null);
            plotFrame.setVisible(true);
        });
    }

    private void showHelp() {
        String helpMessage = """
        ▨ Newton-Raphson Method

        The Newton-Raphson method is an iterative technique for finding approximate roots of a real-valued function.

        Formula:
        x_(n+1) = x_n - f(x_n) / f'(x_n)

        How to use this solver:
        1□Enter the function f(x) in math-like syntax. Example: x^3 - x - 2
        2□Enter the derivative f'(x). Example: 3*x^2 - 1
        3□Provide an initial guess x0 (e.g., 1.5), a tolerance (e.g., 1e-5), and maximum iterations.
        4□Click 'Compute' to see the iteration steps and the root.
        5□The plot will show the function and the root(s) found.
        🔍 Supported functions: sin(x), cos(x), tan(x), log(x), ln(x), exp(x), abs(x), sqrt(x), etc.
        Use ^ for powers (e.g., x^3 for x cubed).
        """;
        JOptionPane.showMessageDialog(this, helpMessage, "Help - Newton-Raphson Method", JOptionPane.INFORMATION_MESSAGE);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(NewtonRaphsonSolver::new);
    }
}
```

REGULA FALSI

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.util.*;
import net.objecthunter.exp4j.*;
import org.jfree.chart.*;
import org.jfree.chart.plot.*;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.xy.*;

public class RegulaFalsiMethodSolver extends JFrame {
    private JTextField functionField, aField, bField, tolField, maxIterField;
    private JTable table;
    private DefaultTableModel tableModel;
    private ArrayList<Double> roots = new ArrayList<>();

    public RegulaFalsiMethodSolver() {
        setTitle("Regula Falsi Method Solver");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(1300, 700);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout(10, 10));

        // Instructions
        JTextArea instructions = new JTextArea(
            "📚 Regula Falsi Method Solver\n" +
            "Instructions:\n" +
            "- Enter f(x) using exp4j syntax (e.g., x^3 - x - 2).\n" +
    "- Supported functions: sin, cos, exp, log, sqrt, etc.\n" +
    "- Use '^' for powers (e.g., x^3).\n" +
    "- Provide interval [a, b], tolerance, and max iterations.\n" +
    "- Ensure f(a) * f(b) < 0 (opposite signs).\n"
);
instructions.setEditable(false);
instructions.setBackground(new Color(240, 240, 240));
instructions.setBorder(BorderFactory.createTitledBorder("Instructions"));
add(instructions, BorderLayout.NORTH);

// Input Panel
JPanel inputPanel = new JPanel(new GridLayout(6, 2, 8, 8));
inputPanel.setBorder(BorderFactory.createTitledBorder("Input Parameters"));
functionField = createField(inputPanel, "Function f(x):", "x^3 - x - 2");
aField = createField(inputPanel, "Interval Start a:", "1.0");
bField = createField(inputPanel, "Interval End b:", "2.0");
tolField = createField(inputPanel, "Tolerance:", "1e-5");
maxIterField = createField(inputPanel, "Max Iterations:", "20");

JButton computeButton = new JButton("Compute");
JButton helpButton = new JButton("Help");

inputPanel.add(computeButton);
inputPanel.add(helpButton);
add(inputPanel, BorderLayout.WEST);

// Table
String[] columns = {"Iter", "xl", "xu", "xr", "Error", "f(xl)", "f(xr)", "f(xu)", "f(xl)*f(xu)"};
tableModel = new DefaultTableModel(columns, 0);
```

```
    table = new JTable(tableModel);
    JScrollPane tableScroll = new JScrollPane(table);
    tableScroll.setBorder(BorderFactory.createTitledBorder("Iteration Table"));
    add(tableScroll, BorderLayout.CENTER);

    // Actions
    computeButton.addActionListener(e -> runRegulaFalsi());
    helpButton.addActionListener(e -> showHelpDialog());

    setVisible(true);
  }

  private JTextField createField(JPanel panel, String label, String defaultText) {
    JLabel lbl = new JLabel(label);
    JTextField field = new JTextField(defaultText);
    panel.add(lbl);
    panel.add(field);
    return field;
  }

  private void runRegulaFalsi() {
    tableModel.setRowCount(0);
    roots.clear();
    try {
      String fxExpr = functionField.getText().trim().replaceAll("\\^", "**"); // Replace ^ with ** for exp4j
      fxExpr = fxExpr.replaceAll("\\*\\*", "^"); // Actually, exp4j uses ^, so keep this!

      double a = Double.parseDouble(aField.getText().trim());
      double b = Double.parseDouble(bField.getText().trim());
```

```
            double tol = Double.parseDouble(tolField.getText().trim());
            int maxIter = Integer.parseInt(maxIterField.getText().trim());

            Expression fx = new ExpressionBuilder(fxExpr).variable("x").build();
            double fa = fx.setVariable("x", a).evaluate();
            double fb = fx.setVariable("x", b).evaluate();

            if (fa * fb >= 0) {
                JOptionPane.showMessageDialog(this, "Error: f(a) and f(b) must have opposite signs.", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            int iter = 1;
            double error = Double.MAX_VALUE;
            double xr = 0;

            while (iter <= maxIter && error > tol) {
                fa = fx.setVariable("x", a).evaluate();
                fb = fx.setVariable("x", b).evaluate();
                xr = b - (fb * (a - b)) / (fa - fb);
                double fxr = fx.setVariable("x", xr).evaluate();
                error = Math.abs(fxr);
                double product = fa * fb;

                tableModel.addRow(new Object[]{
                    iter, String.format("%.8f", a), String.format("%.8f", b), String.format("%.8f", xr),
                    String.format("%.8f", error),
                    String.format("%.8f", fa), String.format("%.8f", fxr), String.format("%.8f", fb),
                    String.format("%.8f", product)
                });

                if (Math.abs(fxr) < tol) break;

                if (fa * fxr < 0) {
                    b = xr;
                } else {
                    a = xr;
                }

                iter++;
            }

            roots.add(xr);
            JOptionPane.showMessageDialog(this, String.format("[OK] Root found at x ≈ %.8f", xr), "Result",
JOptionPane.INFORMATION_MESSAGE);
            showPlot(fx, a, b, roots);

        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        }
    }

    private void showPlot(Expression fx, double a, double b, ArrayList<Double> roots) {
        XYSeries seriesF = new XYSeries("f(x)");

        double xMin = a - 2, xMax = b + 2;
```

```
for (double x = xMin; x <= xMax; x += 0.01) {
    try {
        seriesF.add(x, fx.setVariable("x", x).evaluate());
    } catch (Exception e) {
        // Skip invalid evaluations
    }
}

XYSeriesCollection dataset = new XYSeriesCollection(seriesF);
JFreeChart chart = ChartFactory.createXYLineChart(
    "Regula Falsi Method: Root Finding",
    "x",
    "f(x)",
    dataset,
    PlotOrientation.VERTICAL,
    true,
    true,
    false
);

XYPlot plot = chart.getXYPlot();
plot.setDomainGridlinesVisible(true);
plot.setRangeGridlinesVisible(true);

XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer(true, false);
renderer.setSeriesPaint(0, Color.BLUE);
plot.setRenderer(renderer);

for (double root : roots) {
        XYSeries rootPoint = new XYSeries("Root");
        rootPoint.add(root, 0);
        XYSeriesCollection rootDataset = new XYSeriesCollection(rootPoint);

        XYLineAndShapeRenderer rootRenderer = new XYLineAndShapeRenderer(false, true);
        rootRenderer.setSeriesPaint(0, Color.RED);
        rootRenderer.setSeriesShape(0, new java.awt.geom.Ellipse2D.Double(-5, -5, 10, 10));

        int datasetIndex = plot.getDatasetCount();
        plot.setDataset(datasetIndex, rootDataset);
        plot.setRenderer(datasetIndex, rootRenderer);
    }

    SwingUtilities.invokeLater(() -> {
        JFrame plotFrame = new JFrame("Regula Falsi Plot");
        plotFrame.setSize(900, 600);
        plotFrame.add(new ChartPanel(chart));
        plotFrame.setLocationRelativeTo(null);
        plotFrame.setVisible(true);
    });
}

private void showHelpDialog() {
    String message = """
        🌟 Regula Falsi Method Overview 🌟

        - The Regula Falsi Method is a bracketing technique for root finding.
        - Requires initial guesses a and b such that f(a)*f(b) < 0.
        - Formula: xr = b - fb*(a-b)/(fa-fb)
```

```
            Supported functions: sin, cos, exp, log, sqrt, etc.

            Use '^' for powers (e.g., x^3).


            📌 Example:

            Function: x^3 - x - 2

            a: 1.0

            b: 2.0

            Tolerance: 1e-5

            Max Iterations: 20

            """;

        JOptionPane.showMessageDialog(this, message, "Regula Falsi Method Help", JOptionPane.INFORMATION_MESSAGE);

    }


    public static void main(String[] args) {

        SwingUtilities.invokeLater(RegulaFalsiMethodSolver::new);

    }

}
```