**Plotter Test Plan Document**

**By Aaron Hulseman, Rudy Sanchez, and Sean Clements (Group 16)**

**Table Of Contents**

1. Introduction – p. 1

1.1 Test Plan Objectives – p. 1

1.2 Test Scope – p. 1

1.3 Major Constraints – p. 2

2 Test Plan –p. 2

2.1 Software To Be Tested – p. 2

2.2 Testing Strategy – p. 2

2.2.1 Unit Testing – p. 2

2.2.2 Functionality Testing – p. 2

3 Test Procedure – p. 2

3.1 Software To Be Tested – p. 2

3.2.1 Unit Tests – pp. 2-4

3.2.2 Functionality Tests pp. 4-5

**1. Introduction**

The Plotter programs was given us as a project to help us learn how to test as thoroughly, efficiently, and professionally as possible. Our goal is to eradicate any bugs, errors, or faults that we can by implementing a test plan in the best manner possible.

After testing, this program should be able to accomplish the following:

- Graph functions in Cartesian 2D, Cartesian 3D, and Polar coordinate systems.

- Graph the corresponding derivative function of the one already displayed.

- Calculate the integral of the function, using the Gaussian, Simpson, or Trapezium methods.

- Change the color scheme of the graph or background panel.

- Save the graph as an image file, or export the data in a different format.

- Zoom in or out or center on different portions of the graph.

**1.1 Test Plan Objectives**

The objectives of this test plan are

- To describe what unit tests and integration tests need to be performed.

- To enhance communication between members of the team.

- To set a schedule in place for the completion of these tests.

**1.2 Test Scope**

The unit tests will cover all classes and their dominant methods

Functionality and acceptance testing will cover

- The integral panel and calculations

- Graphing derivatives (for functions that have no derivative as well)

- The functionality of the buttons in the menu bar

- Zoom/centering functions in all coordinate systems.

**1.3 Major Constraints**

- Time is a major constraint on this project. Everything must be completed by 12/3/2016.

- Communication is also a constraint since all members of the team live in different areas of the

country and cannot have any face to face meetings.

- Another constraint is that no team member is an experienced software tester. Some aspects of

testing might be overlooked due to inexperience.

**2. Test Plan**

**2.1 Software To Be Tested**

The software to be tested is the Plotter software.

**2.2 Testing Strategy**

We will use JUnit and ECLemma to unit test each of the classes and their dominant methods and

analyze code coverage of the tests. Afterwards, we will use functionality testing to make sure

the various functions of the program work as expected. Creating and documenting the functionality tests will probably take the most time and effort, so more effort should be applied there. Creating the JUnit tests, on the other hand shouldn’t be too difficult, so minimal effort would be applied here.

**2.2.1 Unit Testing**

The names of the classes to be unit tested are LineData, Point3D, Polygon3D, Renderer3D,

ZBuffer, AdvancedCalculator, Calculator, MathTree, ParseFunction, TNode, and Visualizer. These need to be white-box tested because they are not explored or tested in functionality testing. Most of the focus should be given to Visualizer, Renderer3D, and Calculator and their methods.

**2.2.2 Functionality Testing**

In the functionality testing phase, we will develop a series of tests to demonstrate the

functionality of the software in a black box testing approach. In case of any failure of the

program, we will create a list of defects to be used for creating fixes or future implementations.

This section will test the various interfaces of the system. Namely, the ExportDataPanel, the

IntegralPanel, the Colorpanel, and the main interface.

**3.0 Test Procedure**

**3.1 Software To Be Tested**

The software to be tested with unit tests are the LineData, Point3D, Polygon3D, Renderer3D,

Zbuffer, Visualizer, AdvancedCalculator, Calculator, MathTree, ParseFunction, and TNode

classes. The remaining classes, ColorPanel, DigitTextField, ExportDataPanel, FunctionTextField,

and, IntegralPanel will all be tested with functionality testing. The last class, Main, is a simple

driver which does not need testing.

**3.2.1 Unit Test Cases**

|  |  |  |  |
| --- | --- | --- | --- |
| **TC#** | **Description** | **Input** | **Expected Results** |
| **1** | Calculate Derivative – Trigonometric function | a = 0.0, b = 6.29, n = 4  y = sin(x) | [1, 0, -1, 0] |
| **2** | Calculate Derivative – Logarithmic function | a = 2.0, b = 10.0, n = 4  y = ln(x) | [1/2, 1/4, 1/6, 1/8] |
| **3** | Calculate Derivative – Exponential function | a = 0.0, b = 10.0, n = 5  y = x^2 – 4x | [-4, 0, 4, 8, 12] |
| **4** | Calculate Derivative – Square Root function | a = 2.0, b = 10.0, n = 4  y = sqrt(8x) | [1, sqrt(2)/2, sqrt(2)/sqrt(6), sqrt(2)/sqrt(8)] |
| **5** | Simpson Integral – Trigonometric function | a = 0.0, b = 6.29,  y = cos(x) | .00681464 |
| **6** | Simpson Integral – Logarithmic function | a = 0.0, b = 10.0,  y = e^x | 22025.5 |
| **7** | Simpson Integral – Exponential function | a = 0.0, b = 10.0,  y = 10x -4x^2 | -833.333 |
| **8** | Simpson Integral – Square root function | a = 0.0, b = 10.0,  y = sqrt(3x^3) | 219.089 |
| **9** | Trapezium Integral – Trigonometric function | a = 0.0, b = 6.29,  y = tan(x) | 1.94902 |
| **10** | Trapezium Integral – Logarithmic function | a = 0.0, b = 10.0,  y = 1/ln(x) | 6.16138 |
| **11** | Trapezium Integral – Exponential function | a = 0.0, b = 10.0,  y = 2x^3-x^2 | 4666.67 |
| **12** | Trapezium Integral – Square root function | a = 0.0, b = 10.0,  y = sqrt(x/(x^2 + 4)) | 3.9698 |
| **13** | Gauss Integral – Trigonometric function | a = 0.0, b = 6.29,  y = sin(x) |  |
| **14** | Gauss Integral – Logarithmic function | a = 0.0, b = 10.0,  y = e^(-x) |  |
| **15** | Gauss Integral – Exponential function | a = 0.0, b = 10.0,  y = x^4 – 4x |  |
| **16** | Gauss Integral – Square root function | a = 0.0, b = 10.0,  y = sqrt(x^4 – 4x) |  |
| **17** | Zbuffer – Hex to color conversion | s = “FFFFFF” | r = FF(hex), g = FF(hex), b = FF(hex) |
| **18** | Zbuffer – Color to hex conversion | r = 1, g = B(hex), b = 4 | s = “011104” |
| **19** | Zbuffer – Add zeroes | hexString = “1” | (return) = “01” |
| **20** | Zbuffer – Don’t add zeroes | hexString = “12” | (return) = “12” |
| **21** | LineData – Clone LineData | ldOld = {1, 2, 3, 4} | ldNew = {1, 2, 3, 4} |
| **22** | LineData – Decompose LineData | ld {1, 2, 3, 4} | str = “1,2,3,4” |
| **23** | LineData – Position of function | i = 5, ld = {1, 3, 4, 7} | (return) = -1 |
| **24** | Polygon3D – Add Point Function | xpoints = {}, ypoints = {}, zpoints = {}, npoints = 0  x = 1, y = 1, z = 1 | xpoints = {1}, ypoints = [1], zpoints = [1], npoints = 1 |

**3.2.2 Calculation Functionality Test Cases**

|  |  |  |  |
| --- | --- | --- | --- |
| **TC#** | **Description** | **Execution** | **Expected Results** |
| **1** | Open program | Run Plotter software, or click on executable jar | Program should open |
| **2** | Draw Function | Enter function into bar at top and hit “Draw” | A line should appear that represents given function over the specified integral. Verify that the line is correct for whatever input given. |
| **3** | Draw Derivative | Click on “Show DF” | A different colored line should appear. Verify that the line is correct for the derivative for whatever input is given. |
| **4** | Calculate Simpson Integral | Click on “Integral” select “Simpson” and click “Recalculate” | The integral should be calculated for n = 1000. Verify the results for the function and the interval given. |
| **5** | Calculate Trapezium Integral | Click on “Integral” select “Trapezium” and click “Recalculate” | The integral should be calculated for n = 1000. Verify the results for the function and the interval given. |
| **6** | Calculate Gauss Integral | Click on “Integral” select “Gauss” and click “Recalculate” | The integral should be calculated for n = 1000. Verify the results for the function and the interval given. |
| **7** | Draw 3D plane | Click on 3D Cartesian  Input “x+y” into function text field. Click “draw” | The plane displayed should be a diagonal plane through the origin. |
| **8** | Draw 3D plane translated 1 unit | Click on 3D Cartesian. Input “x+y+1” | The plane will be the same as in test 7 but moved up vertically 1 unit. |
| **9** | Draw polar equation on 3D system | Click on Cartesian 3D. Input “sin(teta)” in function text field. Click “draw” | Should display a sin(x) curve on the graph. |
| **10** | Enter invalid function for 3D system | Click on Cartesian 3D. Input “sin(b)” in function text field. Click “draw” | Program should display error message, saying “Error in parsing function! Please read the readme.txt” |
| **11** | Draw 2D function | From Cartesian 2D system, enter “3x + 1” into function text field and click “draw” | Graph should display a sharply upward diagonal line translated up one unit. |
| **12** | Draw a 3D function on 2D Cartesian | From the Cartesian 2D system, input “x+y” into the function text field and click “draw” | Should receive an error message, or a flat vertical line should appear on the graph. |
| **13** | Draw an invalid function on 2D Cartesian | From the Cartesian 2D system, input “b” into the function text field and click “draw” | Error message should be displayed. |
| **14** | Draw Polar 2D curve | Click on Polar 2D and input “2\*cos(teta)+sin\*(teta)” in the function text field and click “draw” | Should display a circle translated up and to the right of the origin. |
| **15** | Invalid function in Polar 2D | Click on Polar 2D and input “b” into the function text field and click “draw” | Error message should be displayed. |
| **16** | Draw 3D function in Polar 2D | Click on Polar 2D and input “x+y” into the function text field and click “draw” | Function should be displayed since y and x are still variables in the Polar 2D system. |