Regression visualization project document

Tiitus Järvinen (793430)

April 28, 2021

Contents

1 Personal information

Title	Regression UI
Student name	Tiitus Järvinen
Student number	793430
Degree program	Tietotekniikka (Computer Science)
Year	1st
Date	28.4.2021

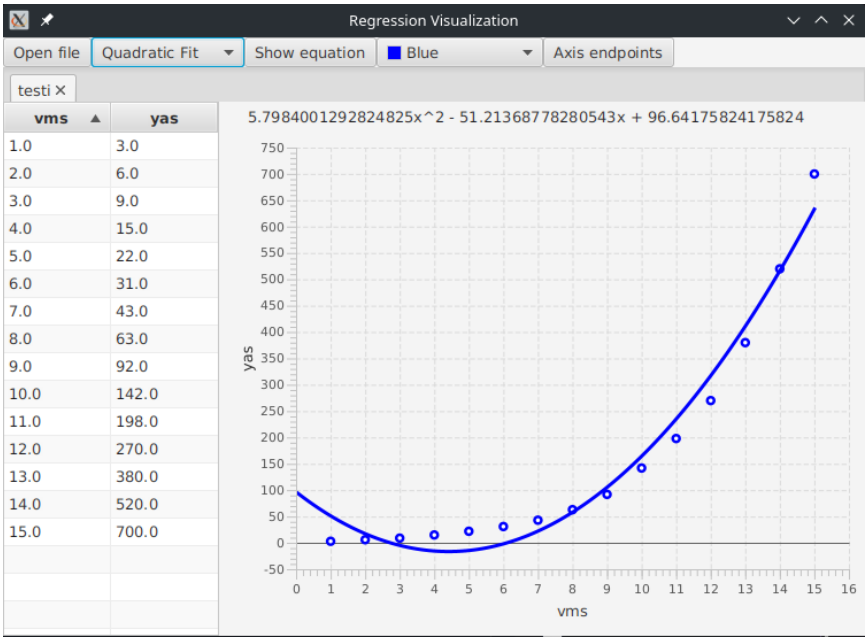
2 General description

The goal of this program is to fit a regression model to a dataset given bythe user. The user should be able to load the dataset from a file or evenfrom a clipboard. The user should be able to configure the appearance ofthe regression fit and coordinates.

The program meets the advanced requirements, with two supported file types (.txt and .json) and two implemented polynomial regression fits (1st and 2nd degree equations).

3 User interface

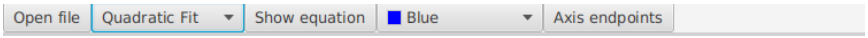
The program is started by building the source code. You can run the program by executing command "sbt run" from the root directory. NOTE: You need java version 55.0 to compile this project (OpenJDK 11).



The UI consists of four different parts:

1. Menubar
2. Data Panel
3. Tab bar
4. Coordinates

3.1 Menubar



The menubar contains the controls of the application.

The Open file-button opens a dialog which the user can use to import data to the software. The Fit-switch button allows the user to choose which fit they want to use with the data. The Show equation-button allows the user to view the equation of the current fit in a selectable text field. The Color switcher allows the user to choose which colour to draw the data points and the fit with. NOTE: The color is black by default for coordinates without a selected regression fit. The Axis endpoints-button allows the user to choose the axis ranges displayed on the coordinates panel. The default "fit into view" option zooms out to show the entire dataset.

3.2 Tab bar

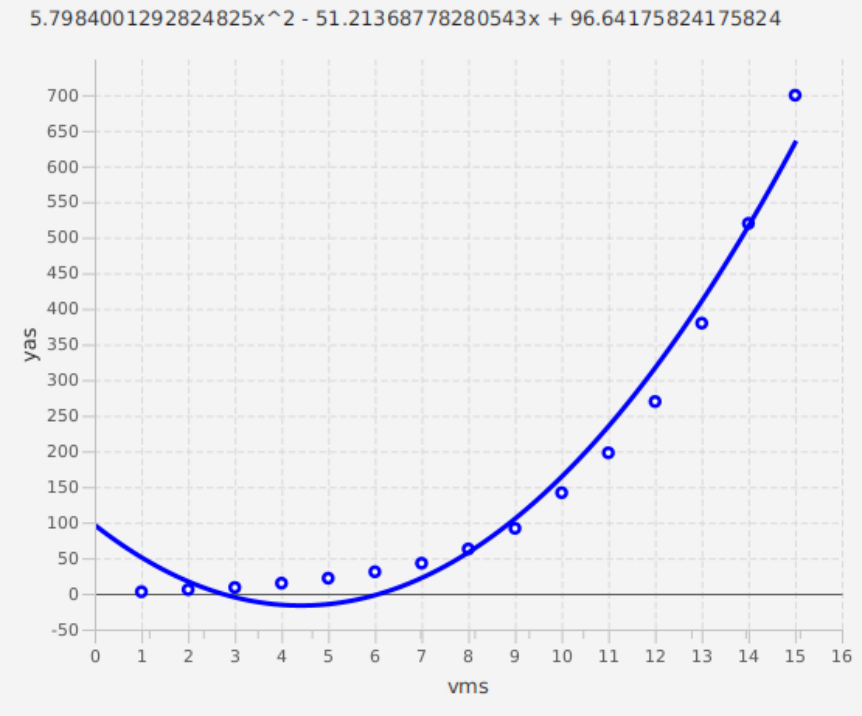
The tabbar allows the user to have multiple datasets open at once, and switch between them.

3.3 Data panel

vms ▲	yas
1.0	3.0
2.0	6.0
3.0	9.0
4.0	15.0
5.0	22.0
6.0	31.0
7.0	43.0
8.0	63.0
9.0	92.0
10.0	142.0
11.0	198.0
12.0	270.0
13.0	380.0
14.0	520.0
15.0	700.0

The data panel allows the user to view the open dataset in an excel-like way. Editing is not allowed though.

3.4 Coordinates



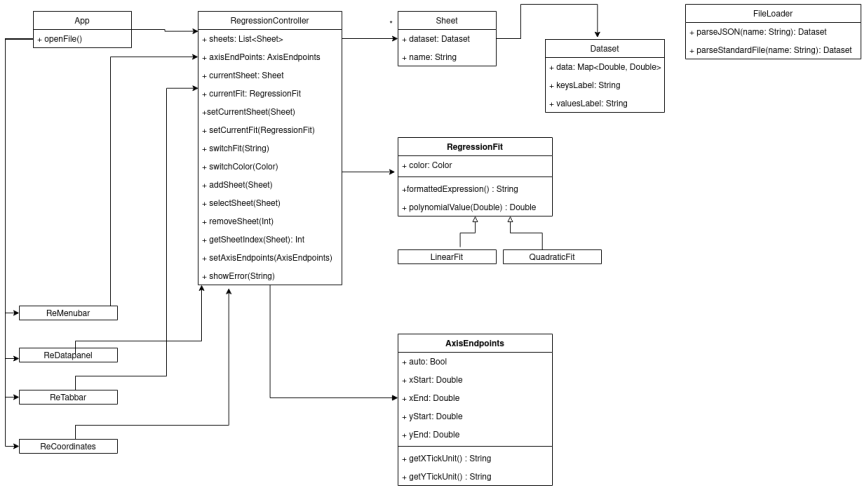
The heart and soul of the program. Displays the given dataset points in a 2D coordinate system, with axis labels. The user can view data of individual points by clicking on them, and choose the drawn regression fit from the menubar.

This component also displays the current fit equation above the coordinates.

4 Program structure

The program UI is composed within the App class and the individual UI components are separated into separate .scala files in the ui subpackage. The UI is controller by the RegressionController class by Observable values, which the UI objects add listeners to as needed.

Here is a UML draft:



The RegressionController instance is passed onto every UI Component that requires it, and when changes are needed the UI components call the functions in the controller, which in turn triggers the listeners of all other UI components.

5 Algorithms

5.1 File loader algorithm

Loads the given file into a Dataset model. The implementation resembles the Chess-program from the OS2 exercises.

5.2 Linear Regression

Uses the Least Squares Method to create a linear scatter plot. Once the slope is calculated, the coordinates object draws it.

The core idea is to minimize the sum of squared differences between the polynomial value of the fit and the data points.

The method as implemented in the program:

```
val n = data.length
val vX = data.map(_._1).reduce((p, p2) => p + p2)/n // AVG of x-values
val vY = data.map(_._2).reduce((p, p2) => p + p2)/n // AVG of y-values
var sXX = 0.0 // Sum of x-AVG(x) squared
var sXY = 0.0 // Sum of (x-AVG(x))(y-AVG(y))
```

```
for (p <- data) {
  sXY += (p._1 - vX) * (p._2 - vY)
  sXX += math.pow(p._1 - vX, 2)
}
val m = sXY/sXX
val b = vY - m*vX
```

Where m is the first coefficient and b is the y-intercept of the fit.

5.3 2nd Degree regression

Uses the Least Squares Method to create the second degree polynomial fit into the given dataset.

Again, the idea is to minimize the sum of squared differences between the polynomial value of the fit and the data points. This time the method is more complex. The coefficients are calculated by using Cramer's law.

```
val Sx        = data.map(_._1).sum // Sum of x-values from the data map
val Sy        = data.map(_._2).sum // Sum of y-values from the data map
val Sx4       = data.map(d => math.pow(d._1, 4)).sum // Sum of x^4
val Sx3       = data.map(d => math.pow(d._1, 3)).sum // Sum of x^3
val Sx2       = data.map(d => math.pow(d._1, 2)).sum // Sum of x^2
val Sxy       = data.map(d => d._1 * d._2).sum       // Sum of x
val Sx2y      = data.map(d => math.pow(d._1, 2) * d._2).sum // Sum of x^2y

val n = data.size // Length of the data

// First coefficient
val coef1 = (Sx2y * (Sx2 * n - Sx * Sx) - Sxy * (Sx3 * n - Sx * Sx2) + Sy * (Sx3 * Sx -

// Second coefficient
val coef2 = (Sx4 * (Sxy * n - Sy * Sx) - Sx3 * (Sx2y * n - Sy * Sx2) + Sx2 * (Sx2y * Sx

// Third coefficient
val coef3 = (Sx4 * (Sx2 * Sy - Sx * Sxy) - Sx3 * (Sx3 * Sy - Sx * Sx2y) + Sx2 * (Sx3 *

(coef1, coef2, coef3)
```

6 Data structures

The RegressionController stores the sheets and current fit and axisendpoints into observable values, which the UI components can add listeners to. The sheets are contained within an ObservableList, whereas the singular objects are contained within ObjectProperties.

The objectproperties are mutable, as is the sheets list.

The Sheet class includes a Dataset class, which will store the points as a Map object. A map object is the most logical solution for connecting two values to each other. In addition, it ensures that the values can be in whatever order. Also, a map structure ensures that no values have the same key, which is essential for the regression analysis part.

Alternatively, a custom data model could have been used, but it would have no visible benefits to a Map object.

7 Files and internet access

The program does not access or require internet in any way.

The program interacts with .txt and .json files. It can open then and parse a set of data points (X-Y) from them. The program does not save the data in any format, nor does it allow for modifying open data in memory, as the purpose of the application is to simply visualize the regression.

Samples of both file types are included in the "example" folder. The following subsections describe the inner structure of these files in case one wants to create them.

7.1 Standard (TXT) format

File saved with a .txt extension. Contains:

1. Dataset X and Y axis labels, separated by a ":"
2. Dataset points in tuple format, with numbers separated by a ":"

Examples can be found in the example data directory. Here is one:

xaxislabel: yaxislabel

1.0 : 20.9

2.1 : 60.4

5.1 : 99.2

7.2 JSON format

File saved with a .json extension. Contains the following format:

```
{
  "keysLabel" : "sample keys label",
  "valuesLabel" : "sample values label",
  "data" : {
    "1.0" : 20.9,
    "2.1" : 60.4,
    "5.1" : 99.2
  }
}
```

8 Testing

The program was tested both automatically and manually during the development process. The testing resembled the planned testing process. By far the biggest testing challenge was the dataset loader part, as it is the most prone to errors. These tests can be ran as regular scalatests.

The UI part of the application was tested manually, by selecting various correct and incorrect values in configuration dialogs and other options.

After adding new features the program's functionality was tested using precreated working datasets and verifying that the program worked as before.

9 Known bugs and missing features

Bugs:

- The standard .txt file format is not parsed correctly if there is an empty line at the end of file. Fixing this should be easy if needed later on.

Missing features:

- You could always have more regression options. Right now I've settled with the two required by the advanced difficulty, but I've structured the data model in a way which allows for easy future expansion.

- The zoom-feature for the coordinate system was not implemented. Currently the user can zoom by selecting the axis ranges manually, but zooming feature with a scrollwheel could be achieved by the same logic as used in the AxisEndpoints method.
- CSV format was not implemented. Would require another method in DatasetLoader class.
- Modifying the data stored in the memory could be a great addition, but not really necessary for the purpose of this program.

10 3 best sides and 3 weaknesses

10.1 Best sides

1. I liked the way I structured the UI with the RegressionController class controlling the UI part of the program. Initially, I had various callback functions passed to the UI components, but now I've minimized those and greatly simplified implementing future features. It also removes a large amount of boilerplate code required by constant callback functions.
2. I prefer the LineChart as provided by the ScalaFX library to the custom implementation I initially wanted to build. The LineChart animates the changes as well!
3. I think the way I build the data model for the different regression fits allows for easy expansion in the future if wanted/needed.

10.2 Weaknesses

- The program misses some functionality that I initially wanted to implement. These are explained at length in the Known bugs and missing features section.
- Lack of automated UI testing. I feel that this could be a problem if this program was developed further, but currently the manual UI testing proved to be satisfactory. There are various UI testing automation libraries that could be used to accomplish this.

11 Derivations from the plan, realized process and schedule

When I created the initial plan, I thought the best way to implement the UI control would be the callback functions passed onto individual components. Instead, I choose to build a controller class similar to those I've used in my personal mobile app projects. Overall, the schedule wasn't a problem at any stage, and especially the regression algorithm implementation proved to be less time consuming, as the calculations were easy to implement by following the documentation in Wikipedia. In contrast, the UI part took a slightly longer time to fully realize.

I didn't originally intend to build the functionality for opening multiple datasets at once, nor did I plan to include the tabbar component. That took a few extra hours.

As I've created various GUI applications in my free time for years, I didn't learn anything particularly new during this process, other than the ScalaFX/JavaFX side of things.

Initially I started with building the test cases and the implementation for loading the two file types. This took roughly the first two weeks. The next weeks I spent creating the general UI layout excluding the coordinates system, which I knew would be the hardest one to implement. After the other parts of the UI were finished, I developed the two regression fits, which took about 1 week, and proceeded to create the coordinates system, which took another week. Finally, I restructured the program to work with a UI controller class and observables.

Overall, the process followed the planned schedule, with slight differences (for example, I didn't intend to redo the UI control part).

12 Final evaluation

I think the final program meets the advanced requirements. The class structure could be improved in certain ways, especially the separation of UI and model logic, and would probably be necessary should the development continue further.

Overall the class structure is well planned to support making changes or future extensions and adding functionality. Adding new regression fits is easy,

and any expansion of UI would be quick to do with the RegressionController implementation.

If I started this process from the start again, I would directly abandon my plans for creating a custom canvas coordinates panel and go straight for the components provided by the ScalaFX library. I would also know that the LaGrange functions are not the proper choice for the 2nd degree polynomial fit.

13 References

ScalaFX and JavaFX documentation. Least Squares Method Cramer's law

14 Appendixes

The code can be found within the src folder.

The example data files are found within the example directory.

Some screenshots of the UI components can be found within the screenshots folder.