1/5/2021

Ali Abdel Razik, Peter Sabry, Maha Ibrahim and Howida Samir

**Nile university**

**Scientific Computing**

Coursework 1 Report

**Scientific Computing**

**January 5, 2021**

|  |  |
| --- | --- |
| *ID* | Name |
| *202002921* | Ali Abdel Razik |
| *18270045* | Peter Sabry |
| *1910273* | Maha Ibrahim Abdel Aleem |
| *1910298* | Howida Samir Mohamed |

Contents

[**Solution Structure** 3](#_Toc65092228)

[**Classes** 3](#_Toc65092229)

[**Functions:** 3](#_Toc65092230)

[**Running the Application:** 6](#_Toc65092231)

[**0.** **Solving linear system using Gauss-elimination and Gauss-Seidel.** 6](#_Toc65092232)

[**1.** **Gauss Elimination - Doing 1-D Regression (with second order polynomial *m = 2*)** 9](#_Toc65092233)

[**1.1** **Plotting for First Dataset** 11](#_Toc65092234)

[**1.2** **Plotting for Second Dataset** 11](#_Toc65092235)

[**2.** **Gauss Elimination - Doing 1-D Regression (with third order polynomial *m = 3*)** 12](#_Toc65092236)

[**2.1** **Plotting for First Dataset** 14](#_Toc65092237)

[**2.2** **Plotting for Second Dataset** 14](#_Toc65092238)

[**3.** **Gauss-Seidel - Doing 1-D Regression (with second order polynomial *m = 2*)** 15](#_Toc65092239)

[**3.1** **Plotting First Dataset** 16](#_Toc65092240)

[**3.2** **Plotting Second Dataset** 16](#_Toc65092241)

[**4.** **Performing 2D Polynomial Linear Regression (Fitting a Plane)** 17](#_Toc65092242)

[**4.1** **Results Summary** 17](#_Toc65092243)

[**4.2** **Using Gauss Elimination Solver** 18](#_Toc65092244)

[**4.2.1** **Using Bedrooms & Bathrooms** 18](#_Toc65092245)

[**4.2.2** **Using Bedrooms & Stories** 19](#_Toc65092246)

[**4.2.3** **Using Bedrooms & Lotsize** 20](#_Toc65092247)

[**4.2.4** **Using Bathrooms & Stories** 21](#_Toc65092248)

[**4.2.5** **Using Bathrooms & Lotsize** 22](#_Toc65092249)

[**4.2.6** **Using Stories & Lotsize** 23](#_Toc65092250)

[**4.3** **Using Gaus-Seidel** 24](#_Toc65092251)

[**4.3.1** **Using Bedrooms & Bathrooms** 24](#_Toc65092252)

[**4.3.2** **Using Bedrooms & Stories** 25](#_Toc65092253)

[**4.3.3** **Using Bedrooms & Lotsize** 26](#_Toc65092254)

[**4.3.4** **Using Bathrooms & Stories** 27](#_Toc65092255)

[**4.3.5** **Using Bathrooms & Lotsize** 28](#_Toc65092256)

[**4.3.6** **Using Stories & Lotsize** 29](#_Toc65092257)

[**5.** **Performing Interpolation** 31](#_Toc65092258)

# **Solution Structure**

Environment Used: Visual Studio 2019 – C++ Project

Source Code: <https://github.com/PeterSabry84/Numerical_Solver.git>

Notes:

* The code did not use any third-party libraries (not even STL). We have implemented our own class of dynamically allocated matrix.
* The code used heap memory allocation for efficiency. Use of scoped variables (in the stack memory) was minimal especially for large objects.
* For efficiency, functions arguments were passed by reference (or a pointer was passed).
* Several constructors per class were implemented to allow for easier, more intuitive interface.
* Copy assignment operators, copy constructors and destructors were implemented. (as object’s internal states -member variables- are dynamically allocated in the heap for efficient memory management).
* Naming conventions were followed for as much as possible.
  + Class names start with capital letter.
  + Identifiers have meaningful names.
  + Etc.
* Definition files -.h header files- that show the function interface, were separated from the implementation files (.CPP).

## **Classes**

|  |  |
| --- | --- |
| *Class Name* | Main Purpose |
| *Linear\_system* | Solving linear equations  (Using Gauss Scaled Elimination or/ Gauss-Siedel Iterations) |
| *univar\_regressor* | Doing 1-D linear regression (Line Fitting or/ Curve Fitting) |
| *multivar\_regressor* | Doing 2-D linear regression (Plane Fitting) |
| *Matrix* | Facilitate using 2D arrays |
| *Newton\_interpolator* | Performing Newton Interpolation |
| *Spline\_Interpolator* | Performing Spline Interpolation |

## **Functions:**

|  |  |  |
| --- | --- | --- |
| *Class Name* | Function Name | Main Purpose |
| *Linear\_system* | Constructor:  Linear\_system(Matrix A, Matrix b) |  |
| Solve() | * Solve linear equations using Gauss Scaled Elimination * Uses two variables of type Matrix: A, b, both are forming an augmented matrix that is used to fined the solution |
| solve\_iteratively(double initials[], const int& n\_iter) | * Solve linear equations using Gauss-Siedel iterations, it accepts initial values and number of iterations |
|  |  |
| bool is\_valid\_solution() | * Returns a Boolean reflecting the ill-condition state of the system of equations |
| *univar\_regressor* | Constructor  univar\_regressor(const double x[], const double y[], const int m, const int s) | Accepts X, Y as array data points  *m* is the desired polynomial degree  *s* is an option to use gauss elimination or gauss-siedel iterations |
| fit() | Does the regression task on the points x, y   * Computes the Augmented Matrix (A| b) * Calls the Linear\_system.solve() or Linear\_system.solve\_iteratively() to get solution * Returns the solution |
| predict(const double xi, const Matrix coeff, const int m) | Makes a single prediction for a point x  Called within a loop to predict multiple points |
| *multivar\_regressor* | multivar\_regressor(const Matrix x, const double y[], const int n, const int m) | Almost the same as univar\_regressor except for the input X is of type matrix to accommodate for x[n, 2] input |
| fit() | Same as in univar\_regressor, difference in the way the coefficients matrix is computed |
| predict(const double xi[], const Matrix coeff, const int m) | Same as in univar\_regressor |
| *Newton\_interpolator* | Constructor  Newton\_interpolator(double\* x, double\* y, int size) |  |
| finite\_difference(const int& first, const int& last) | Computes finite differences |
| fit() | Uses finite\_difference to fit a polynomial |
| interpolate(const double& x\_new) | Interpolates a point |
|  |  |  |
| *Spline\_Interpolator* | Constructor  Spline\_interpolator(double\* x, double\* y, int size) |  |
| fit() |  |
| interpolate(const double& x\_new) |  |

# **Running the Application:**

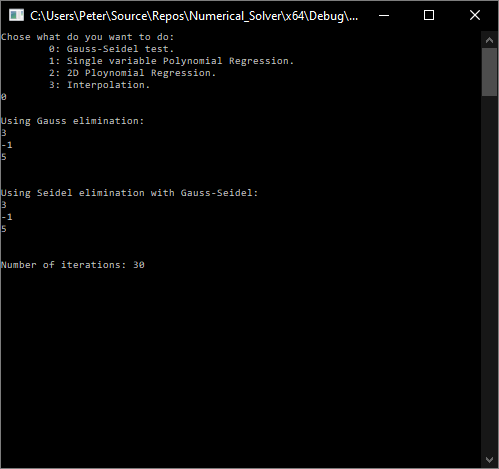
The binaries code is a simple command prompt interface. It starts off with the selection menu:

Selection of 0, runs the test case below using three function calls:

1. Linear\_system.solve()



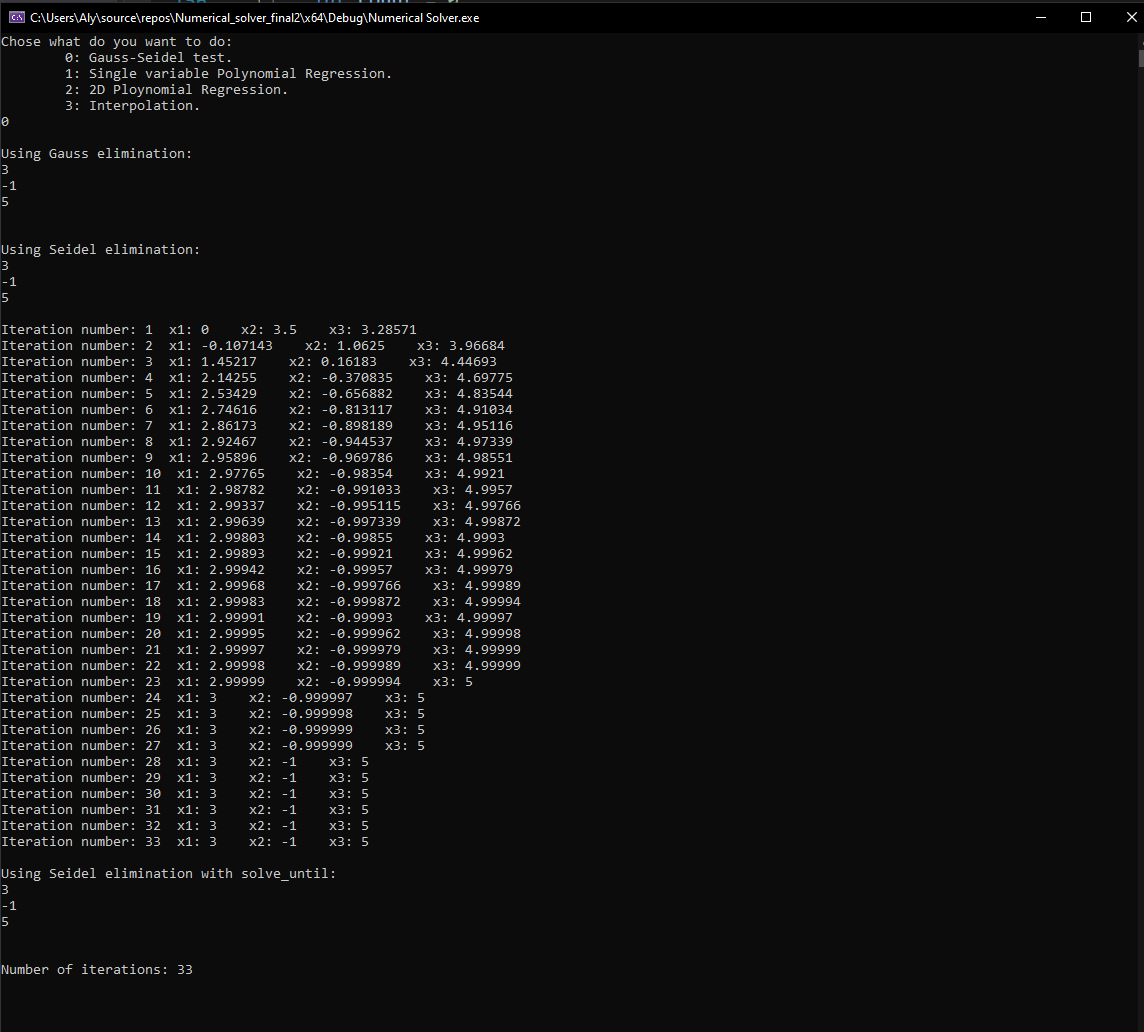
## **Solving linear system using Gauss-elimination and Gauss-Seidel.**



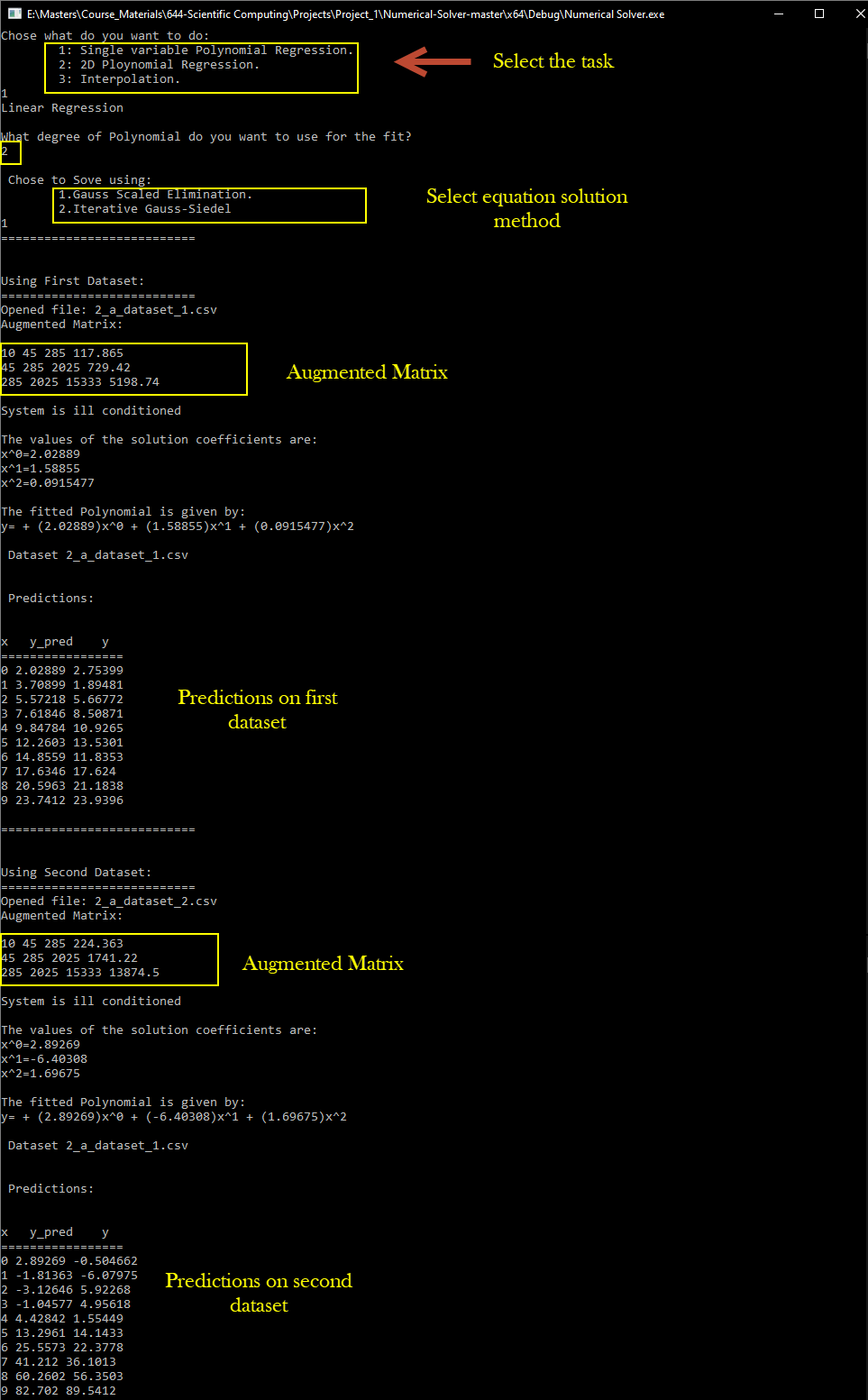
* The first solve() function call, uses Gauss elimination.
* The solve\_until() function call, uses Gauss-Seidel. The function signature takes arguments to set criterion for termination, specifying a tolerable error.

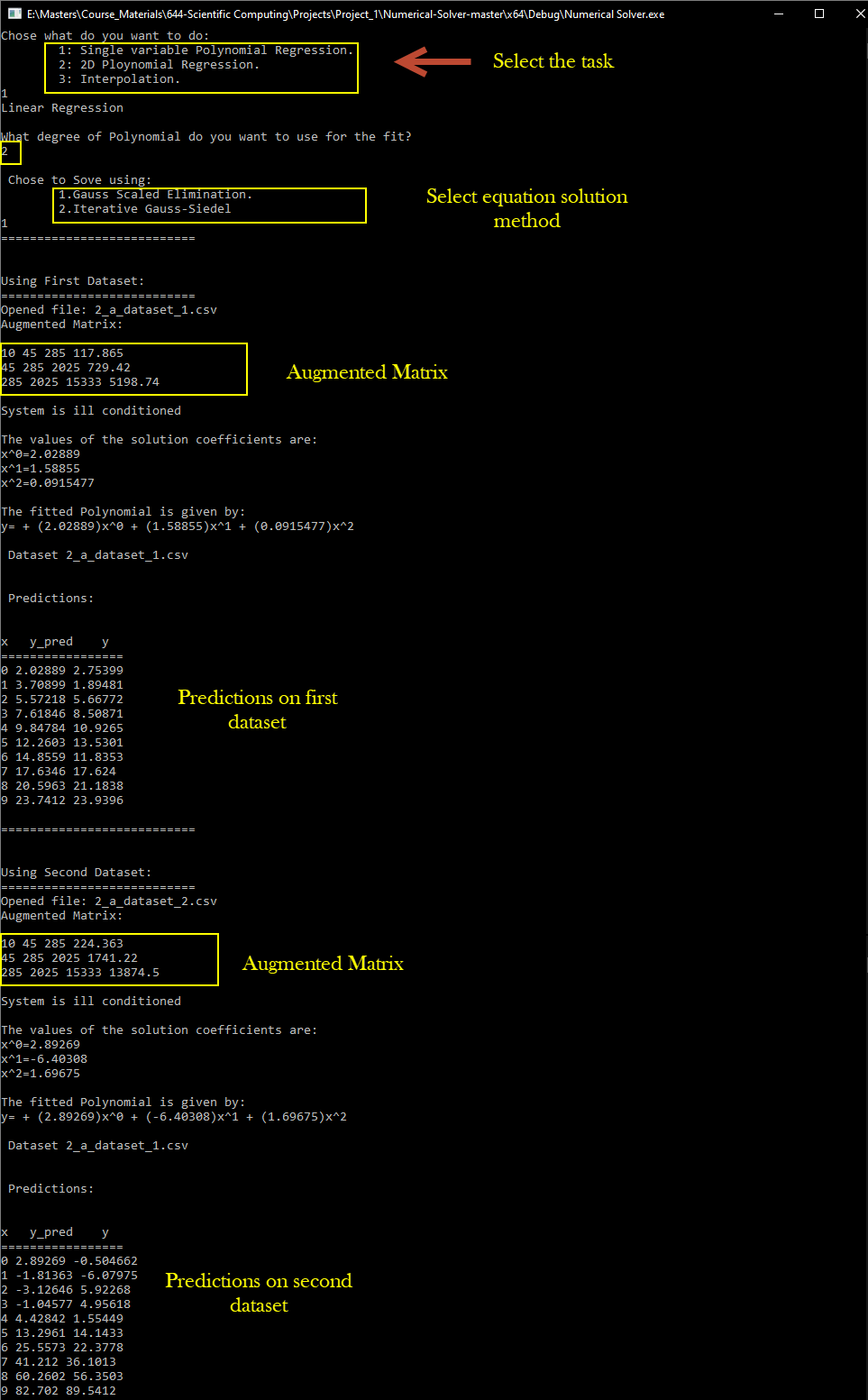
**B. Gauss Seidel:**

Showing the first 30 iterations and values of variables as they change through iterations:

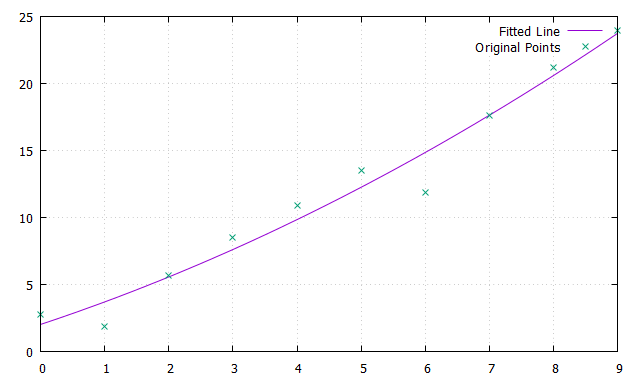


## **Gauss Elimination - Doing 1-D Regression (with second order polynomial *m = 2*)**

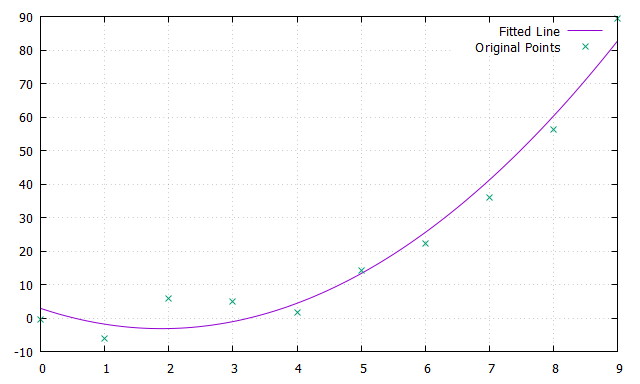




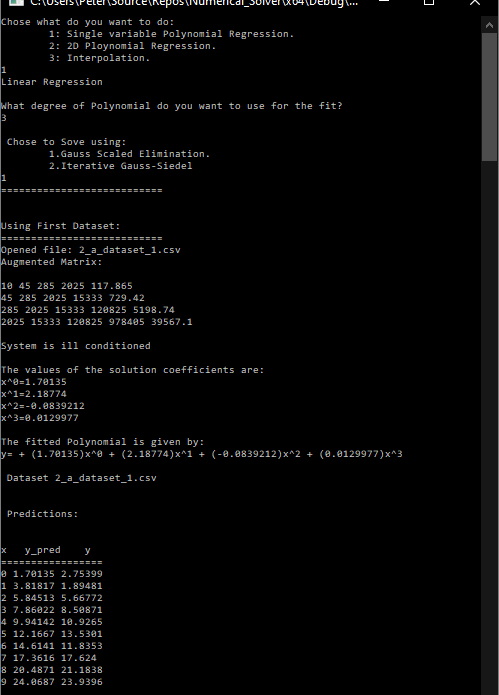
### **Plotting for First Dataset**

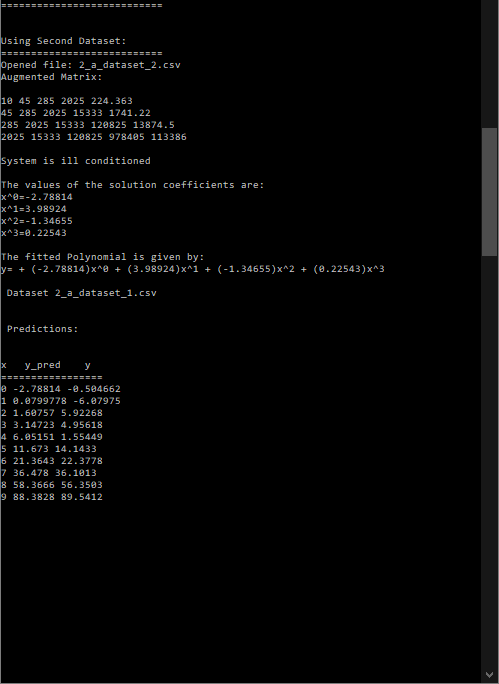


### **Plotting for Second Dataset**

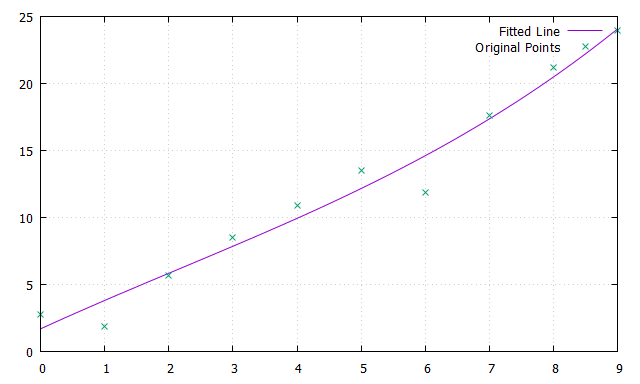


## **Gauss Elimination - Doing 1-D Regression (with third order polynomial *m = 3*)**

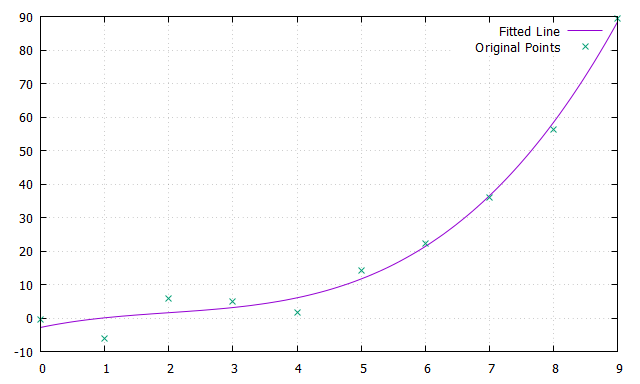




### **Plotting for First Dataset**



### **Plotting for Second Dataset**

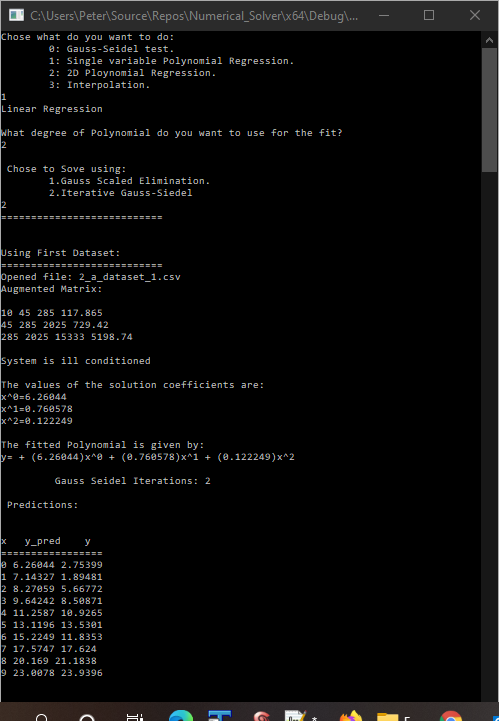
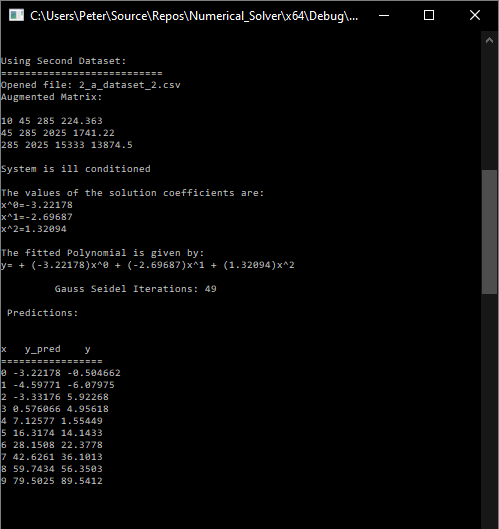


**It is clear that a third order polynomial fits better than a second order**

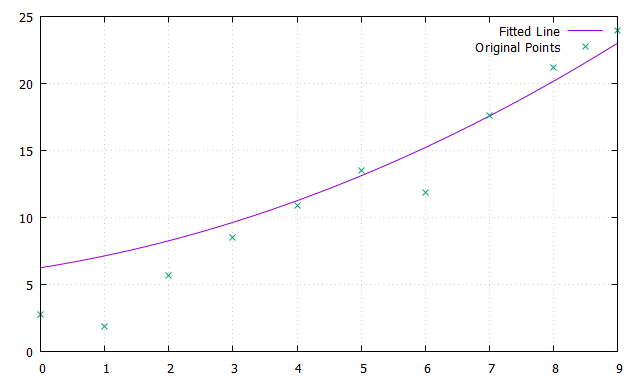
## **Gauss-Seidel - Doing 1-D Regression (with second order polynomial *m = 2*)**

***Number of iterations:***

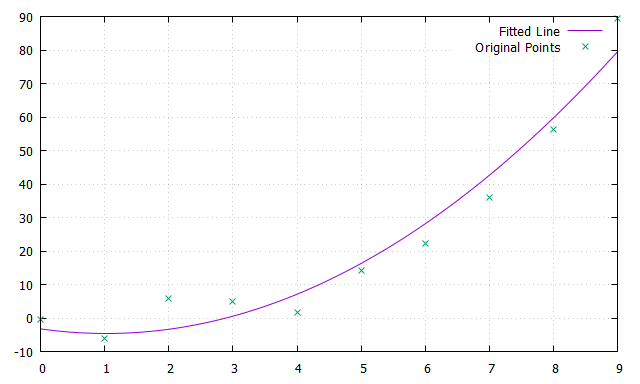
* **First Dataset: 2**
* **Second Dataset: 49**

### **Plotting First Dataset**



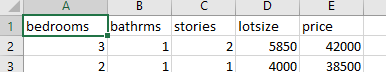
### **Plotting Second Dataset**



## **Performing 2D Polynomial Linear Regression (Fitting a Plane)**

**Notes/ Assumptions:**

* The Housing.csv file was edited by removing the un-needed columns and keeping only the following:



* All columns are in different scales, normalizing was used according to the following simple form: normalized\_value = (original\_value – min\_value) / (max\_value – min\_value)
* Mean Square Error was calculated based on normalized values

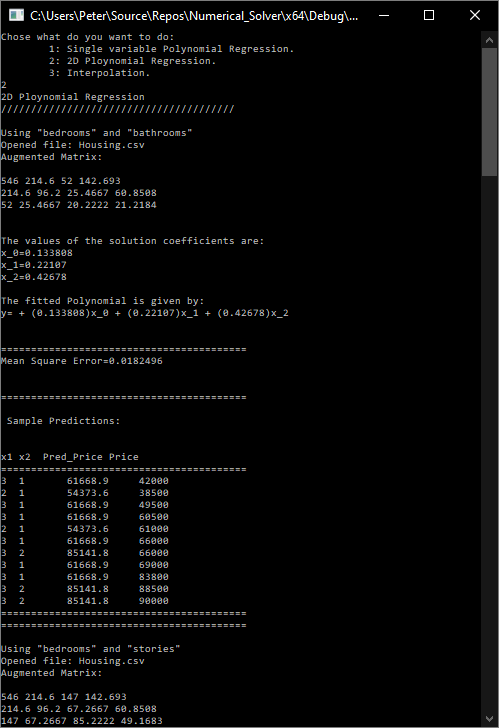
### **Results Summary**

* It was noticed that MSE is the lowest when using "**bathrooms**" and "**lotsize**" features as per the following summary table.

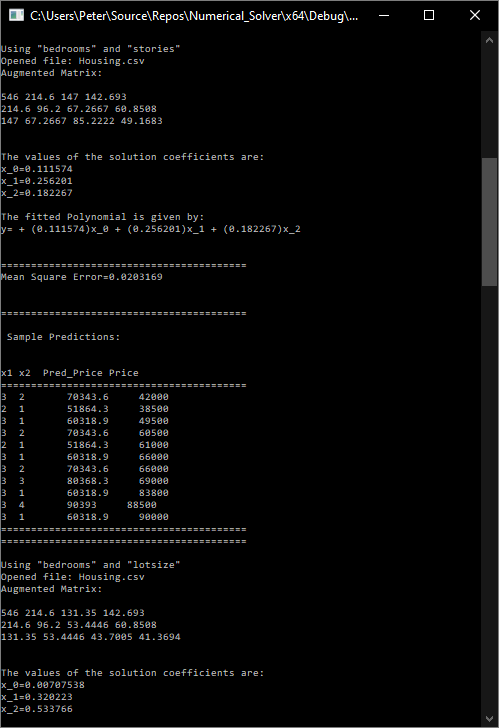
|  |  |  |
| --- | --- | --- |
| **Fearures** | **Gauss Elimination MSE** | **Gauss-Seidel MSE** |
| Using "bedrooms" and "bathrooms" | 0.01825 | 0.01832 |
| Using "bedrooms" and "stories" | 0.020317 | 0.02050 |
| Using "bedrooms" and "lotsize" | 0.016463 | 0.016554 |
| Using "bathrooms" and "stories" | 0.017282 | 0.018348 |
| **Using "bathrooms" and "lotsize"** | **0.014007** | **0.014018** |
| Using "stories" and "lotsize" | 0.014909 | 0.014920 |

### **Using Gauss Elimination Solver**

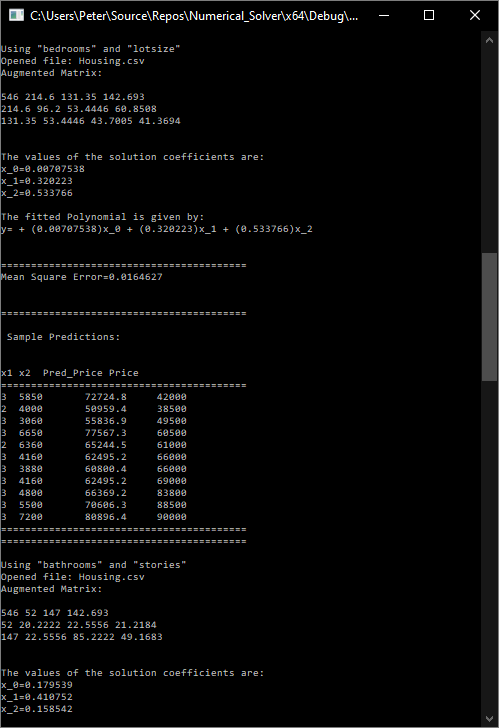
### **Using Bedrooms & Bathrooms**



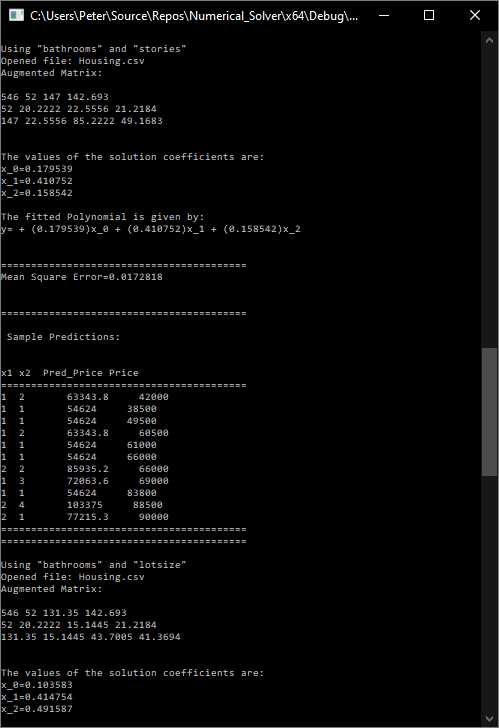
### **Using Bedrooms & Stories**



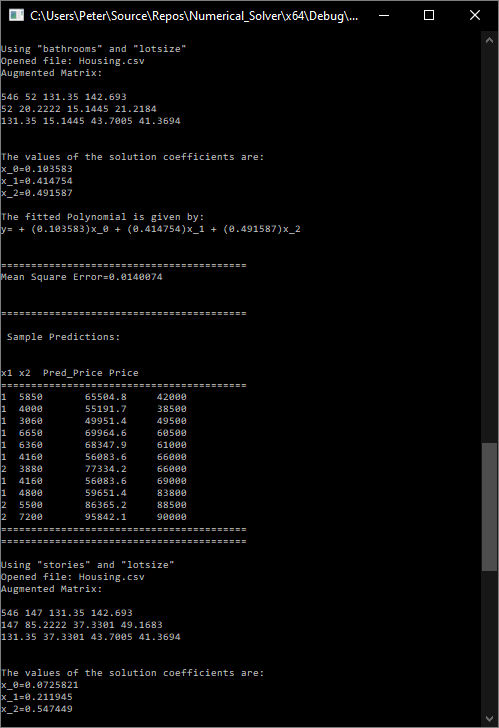
### **Using Bedrooms & Lotsize**



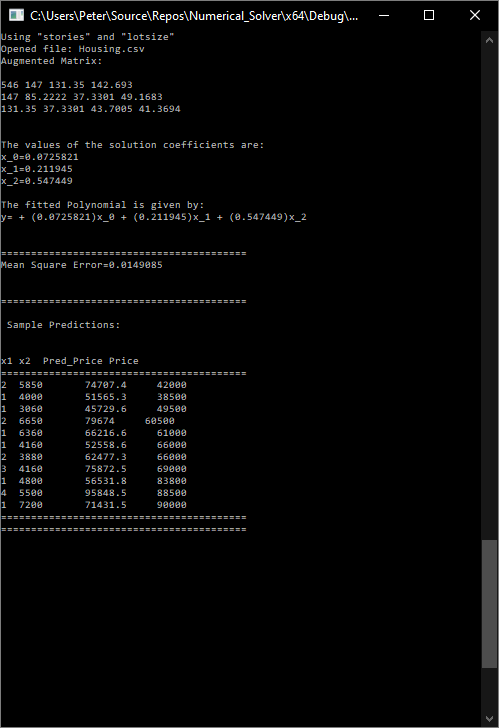
### **Using Bathrooms & Stories**



### **Using Bathrooms & Lotsize**

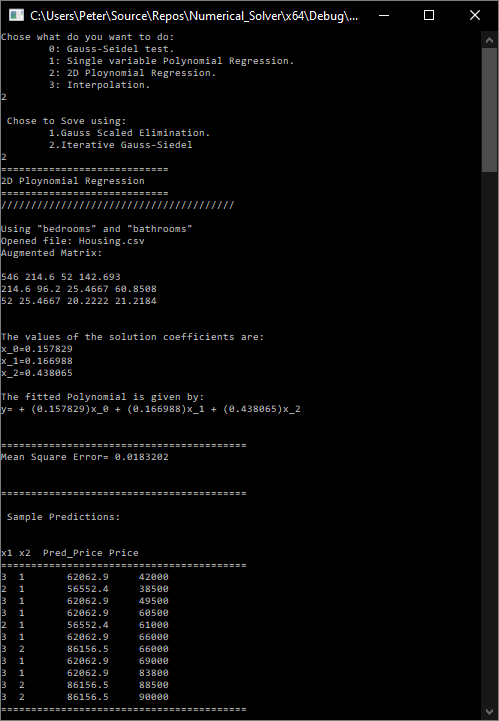


### **Using Stories & Lotsize**

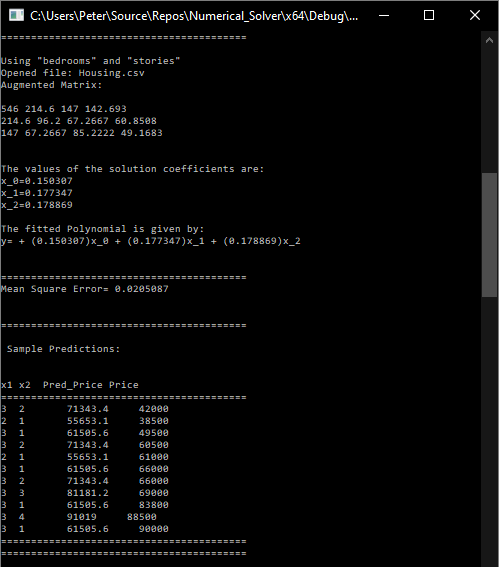


### **Using Gaus-Seidel**

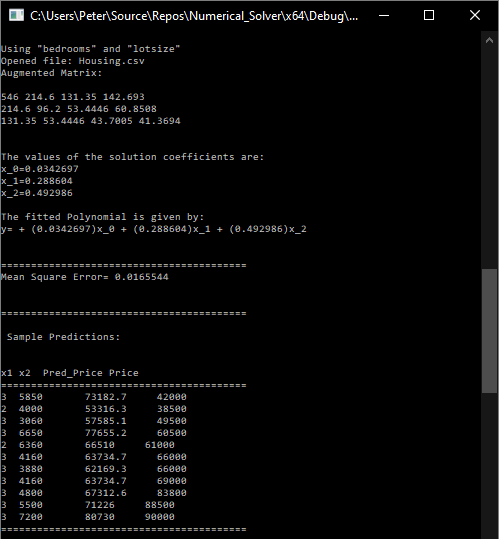
### **Using Bedrooms & Bathrooms**



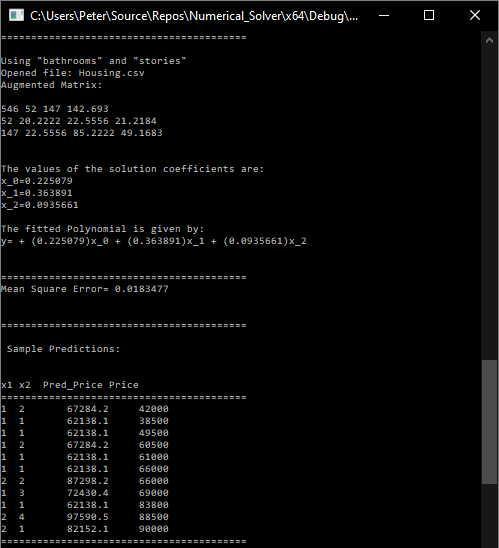
### **Using Bedrooms & Stories**



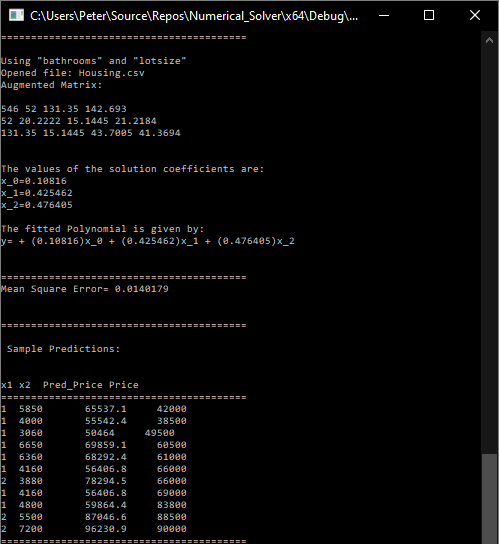
### **Using Bedrooms & Lotsize**



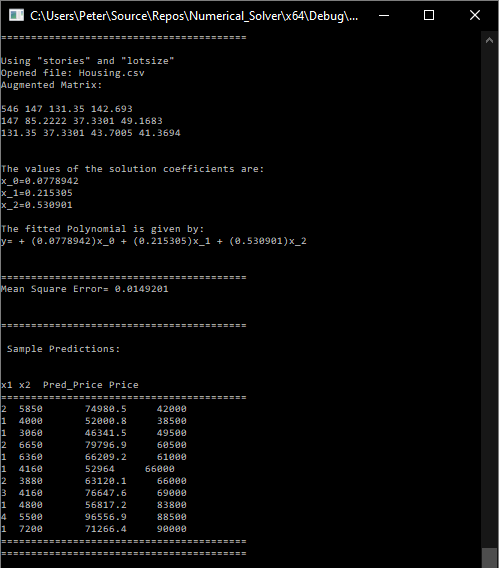
### **Using Bathrooms & Stories**



### **Using Bathrooms & Lotsize**



### **Using Stories & Lotsize**



The number of iterations for Seidel was:

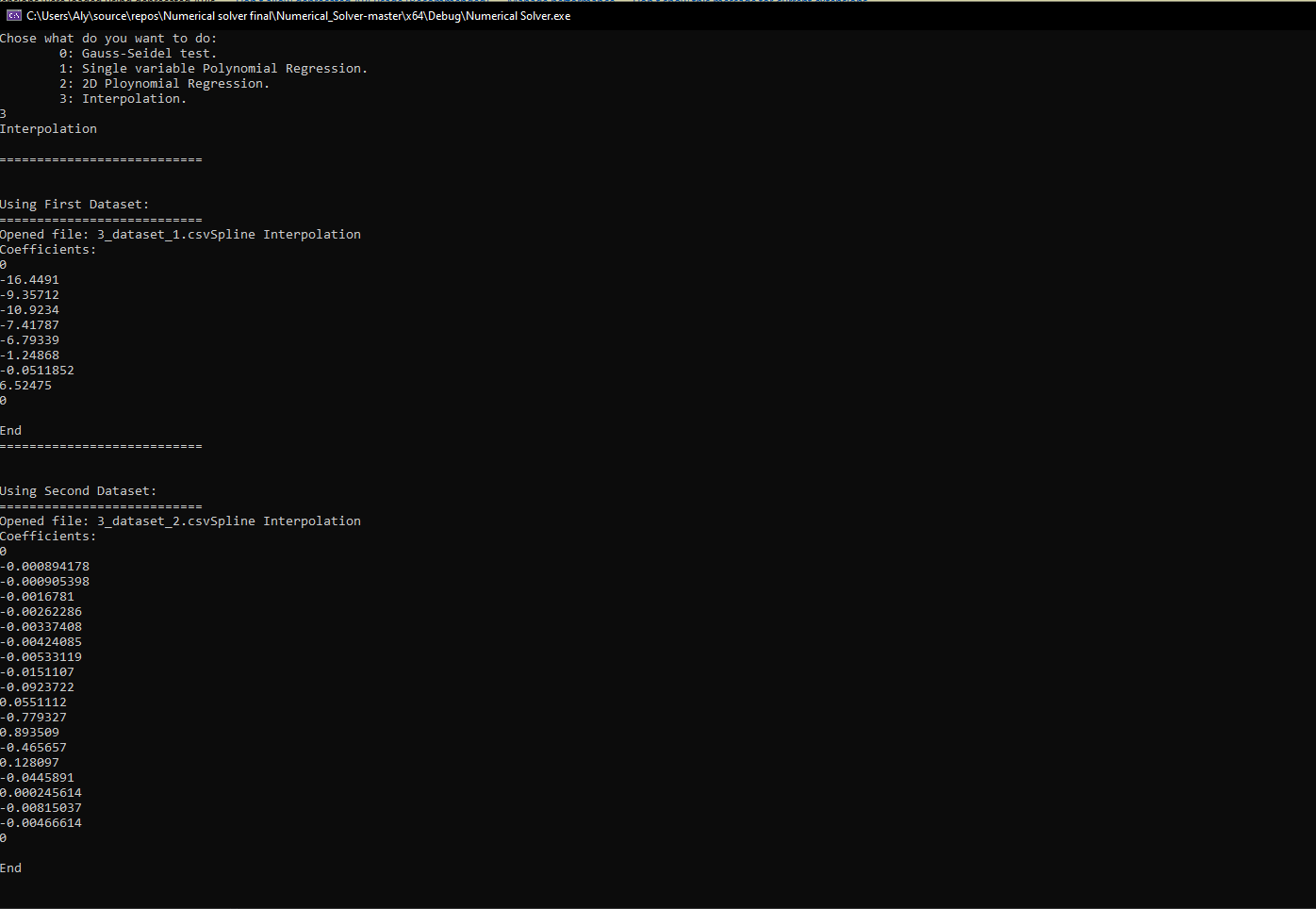
**Convergence criterion:**

For each row in the matrix, the absolute value of the diagonal element must be greater than the sum of the absolute values of the off‐diagonal elements.

We set the 0.01 as the allowed delta change.

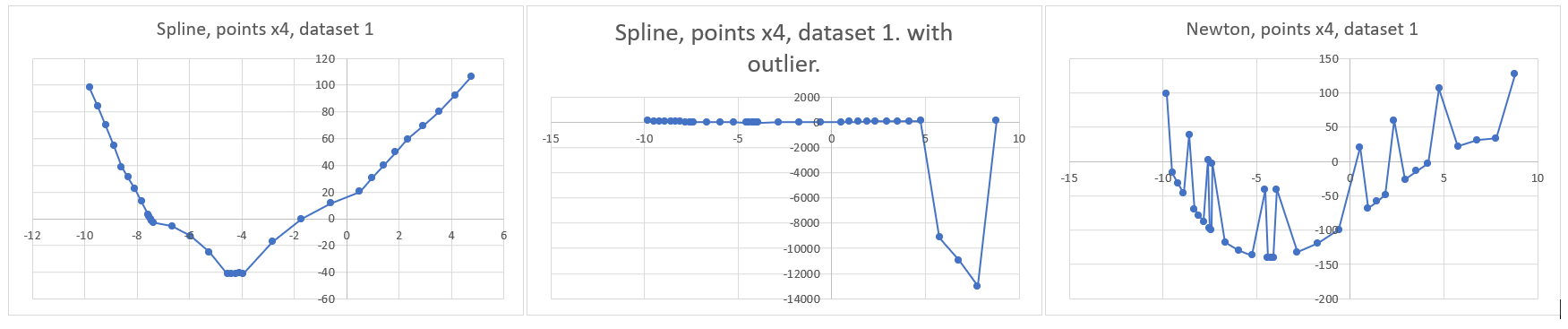
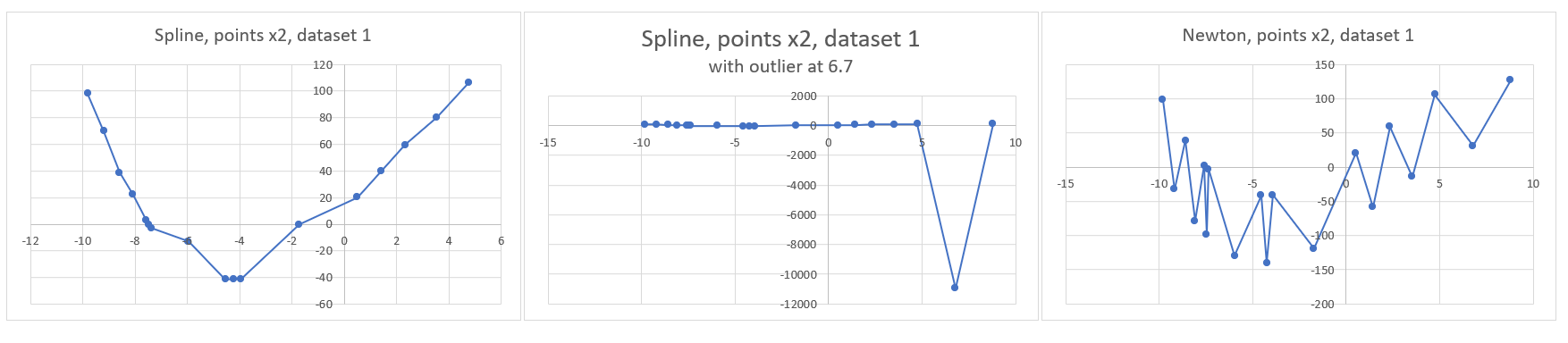
## **Performing Interpolation**

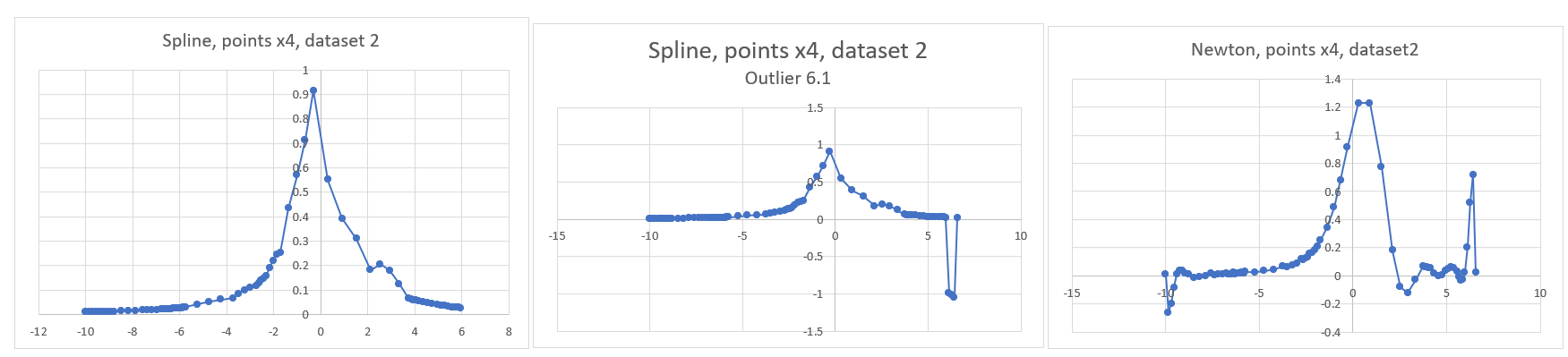
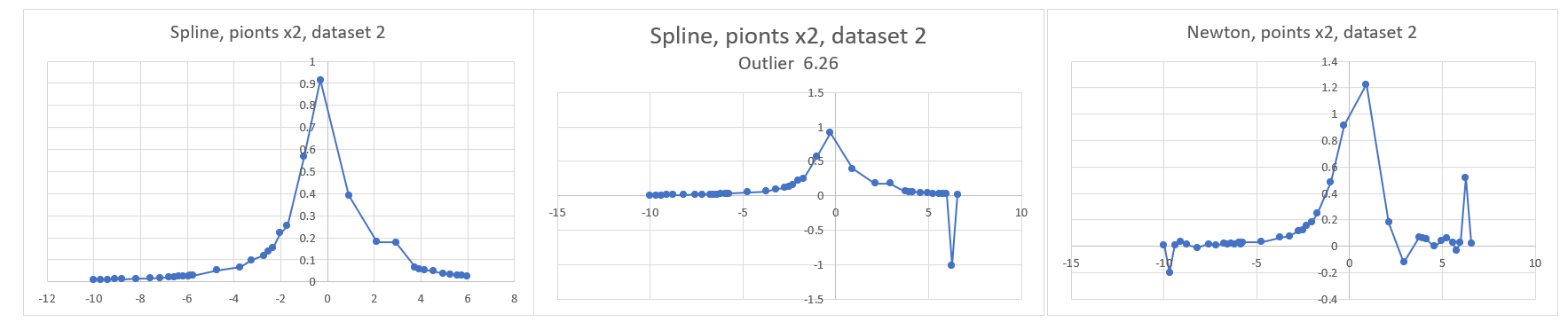
**Coefficients are saved in .csv file for both cases (newton and splines). Below is a screen print of coefficients in the case of splines.**



**Charts:**

**Please find below charts as required. Please refer to chart title.**





**Remarks:**

* In these plots, interpolated dots are connected by linear segments.
* As we double and quadruple the interpolated points, the underlying trend should become evident.
* In dataset 1, it is seen that newton interpolation experience jerky interpolation. Spline interpolation seems to capture the trend better and is smooth.
* Higher orders are required to fit datasets with multiple points in Newton method. This introduces round-off errors and overshoot. Cubic splines method uses only 3rd degree polynomials and connect the segments. The cubic splines method here approximates abrupt changes better than Newton method (smooth).