**Project 1: Scientific Computing**

January 3, 2021

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

# **Solution Structure**

Environment Used: Visual Studio 2019 – C++ Project

## **Classes**

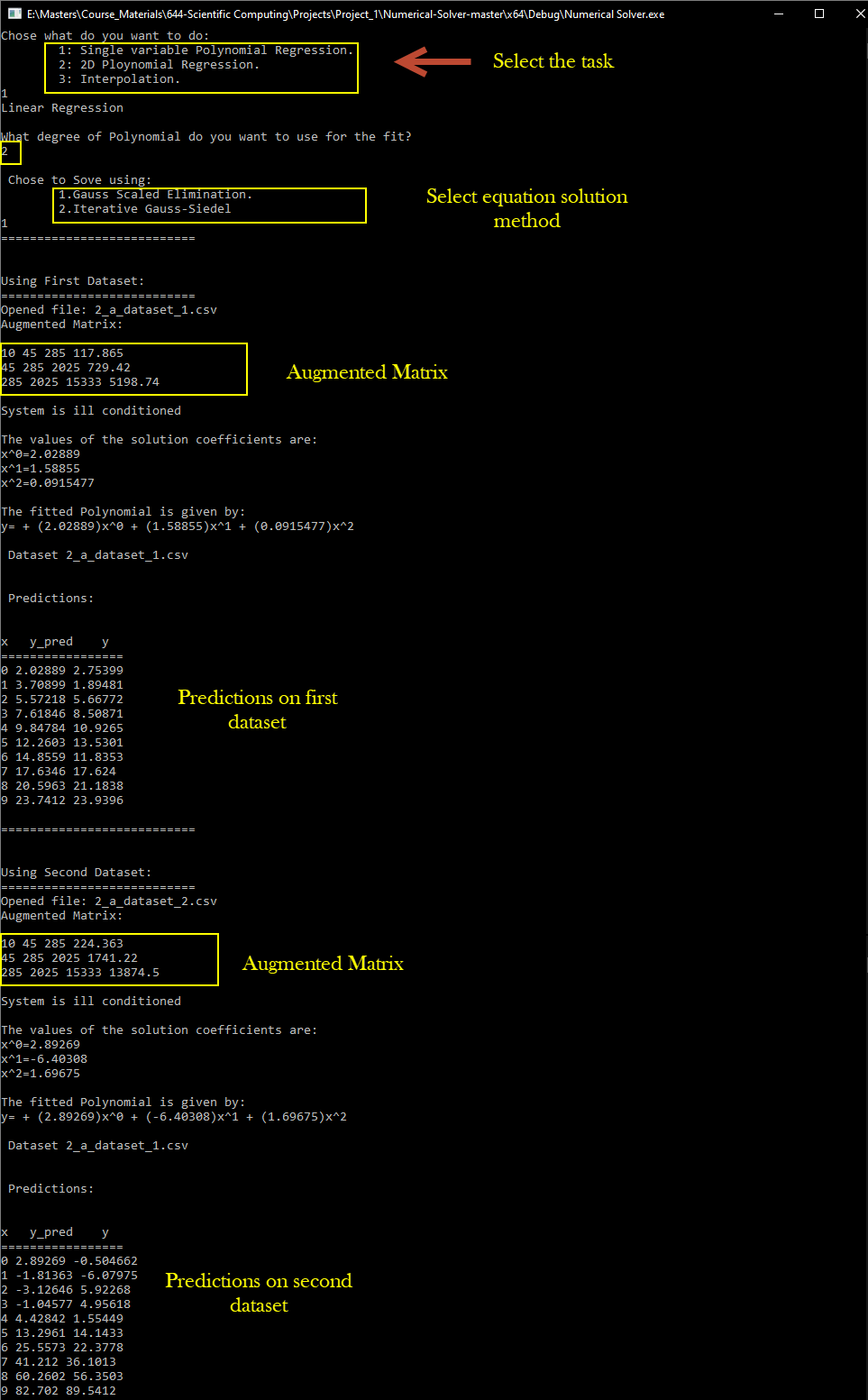
|  |  |
| --- | --- |
| *Class Name* | Main Purpose |
| *Linear\_system* | Solving linear equations  (Using Gauss Scaled Elimination or/ Gauss-Siedel Iterations) |
| *univar\_regressor* | Doing 1-D linear regression (Line Fitting or/ Curve Fitting) |
| *multivar\_regressor* | Doing 2-D linear regression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

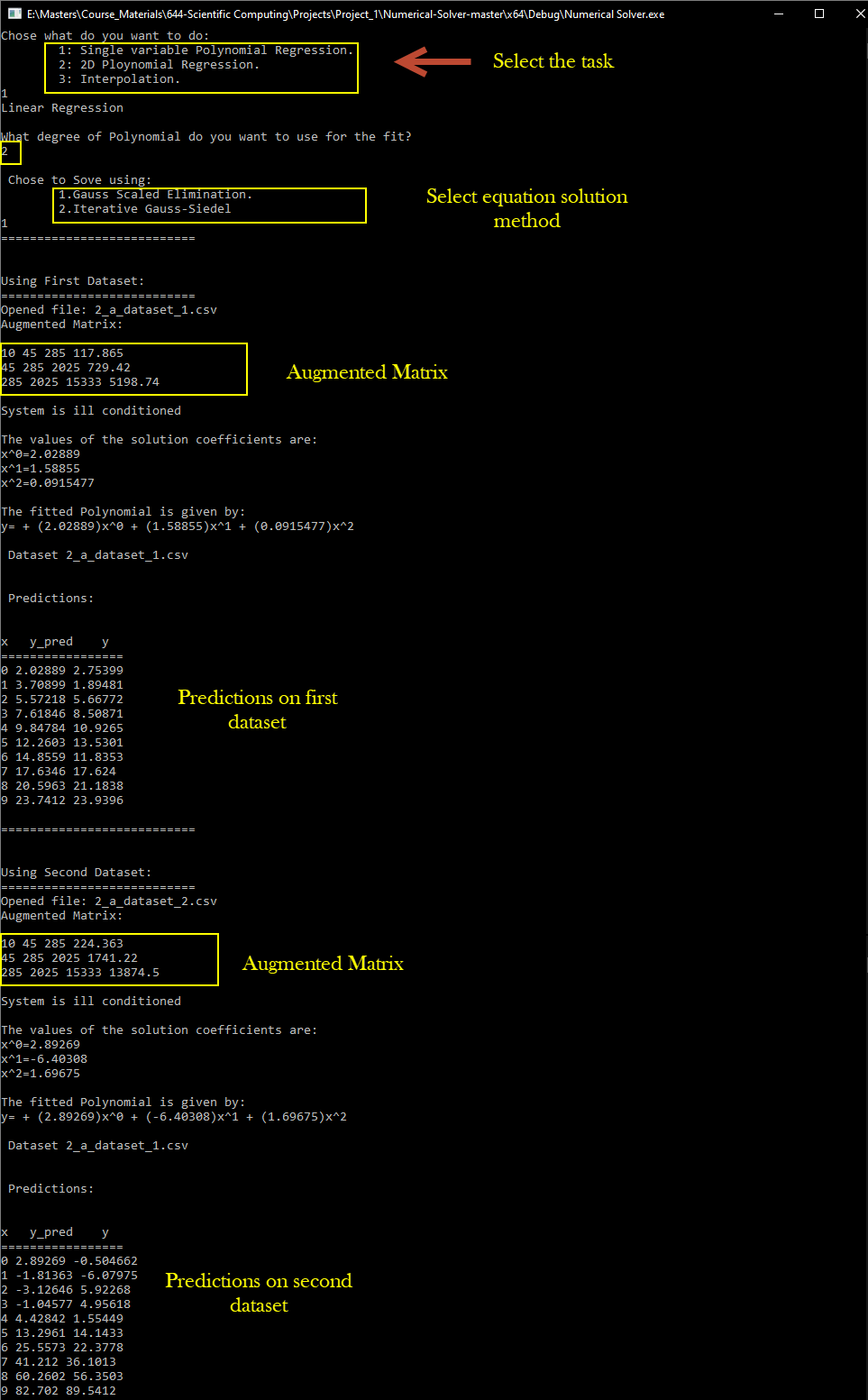
## **Functions:**

|  |  |  |
| --- | --- | --- |
| *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) |  |

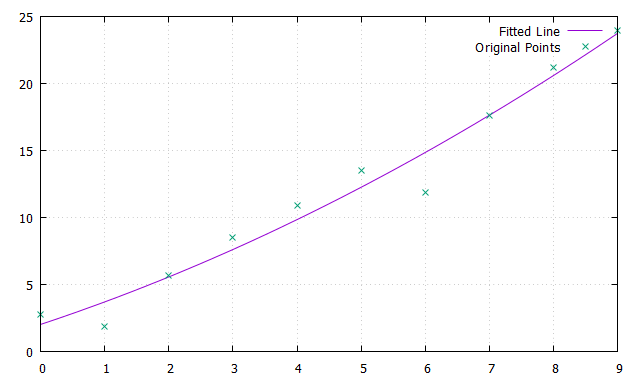
# **Running the Application:**

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

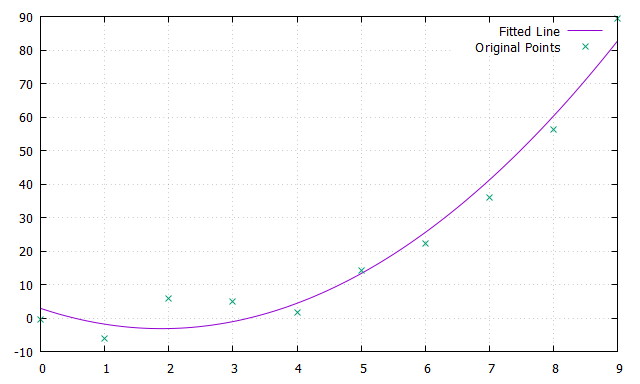




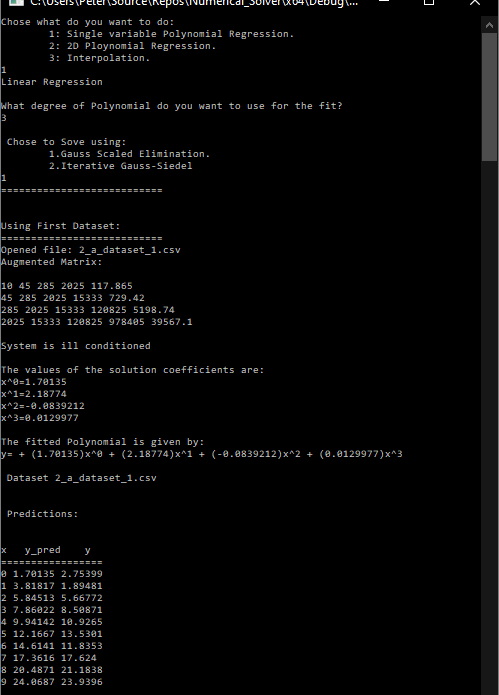
### **Plotting for First Dataset**

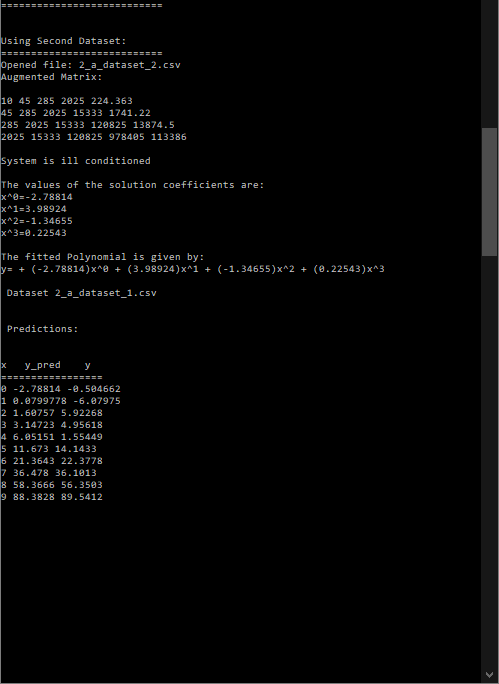


### **Plotting for Second Dataset**

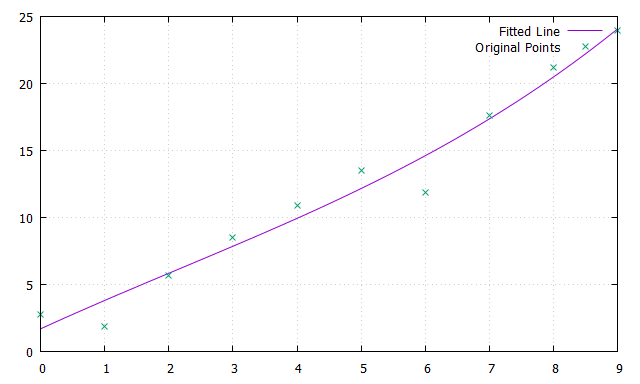


## **Doing 1-D Regression (with third order polynomial *m = 3*)**

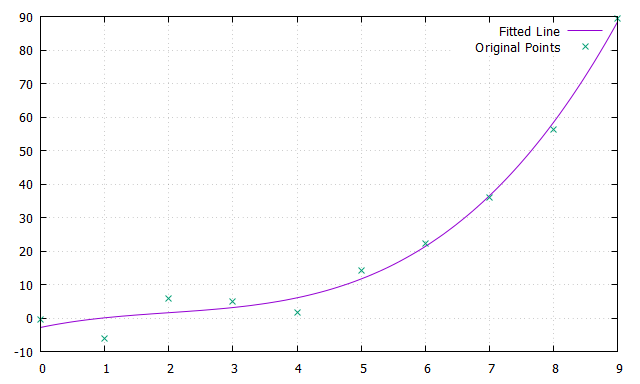




### **Plotting for First Dataset**



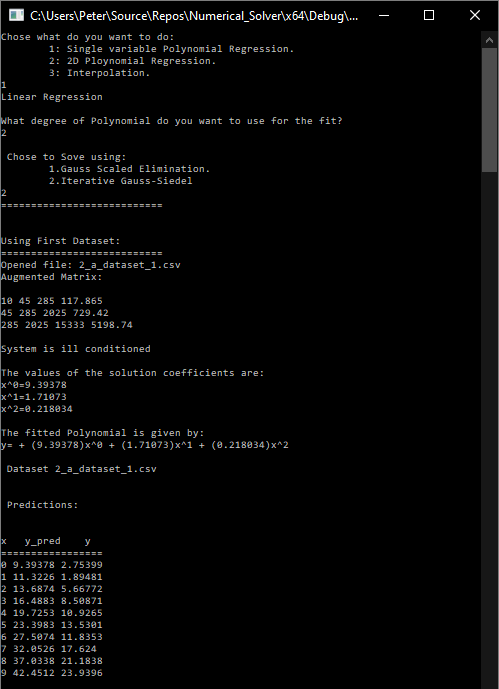
### **Plotting for Second Dataset**

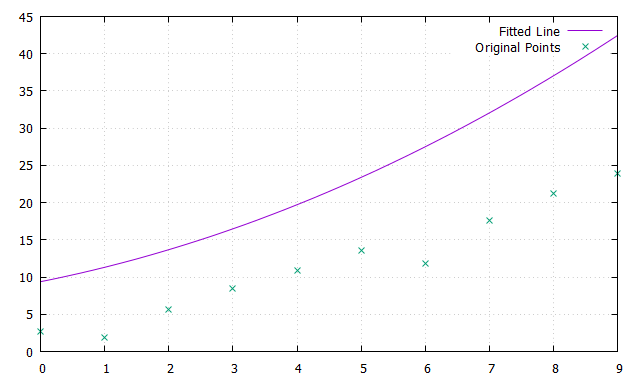


**It is clear that a third order polynomial fits better than a second order**

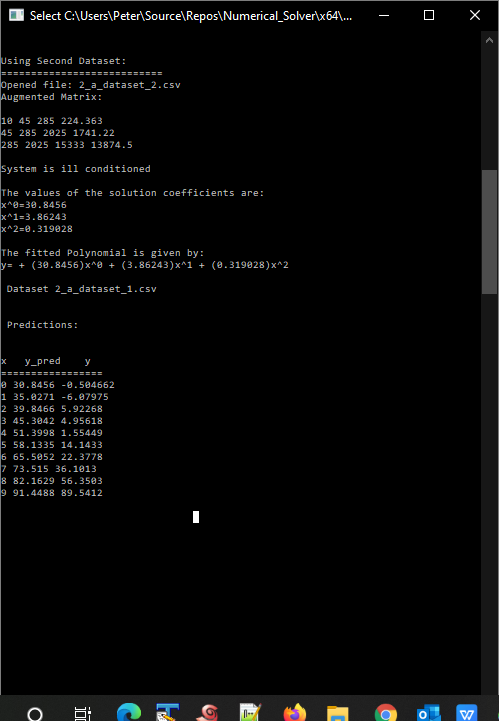
## **Doing 1-D Regression (with second order polynomial *m = 2*) / Gauss-Siedel**

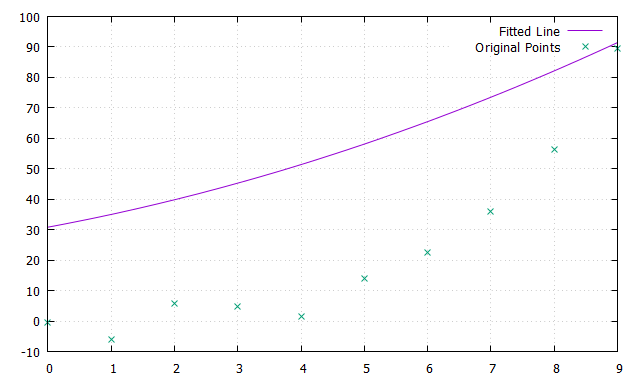
### **Testing on First Dataset**





### **Testing on Second Dataset**

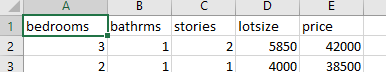




## **Performing 2D Polynomial Linear Regression (Fitting a Plane)**

**Notes/ Assumptions:**

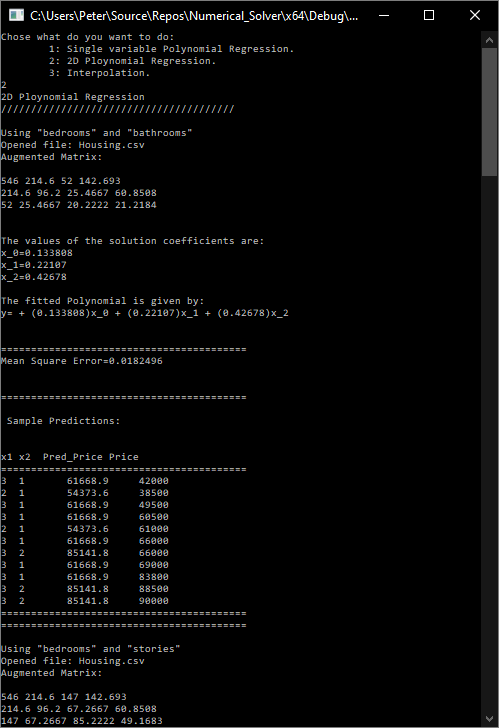
* The Housing.csv file was edited by removing the un-needed columns and keeping only the following:



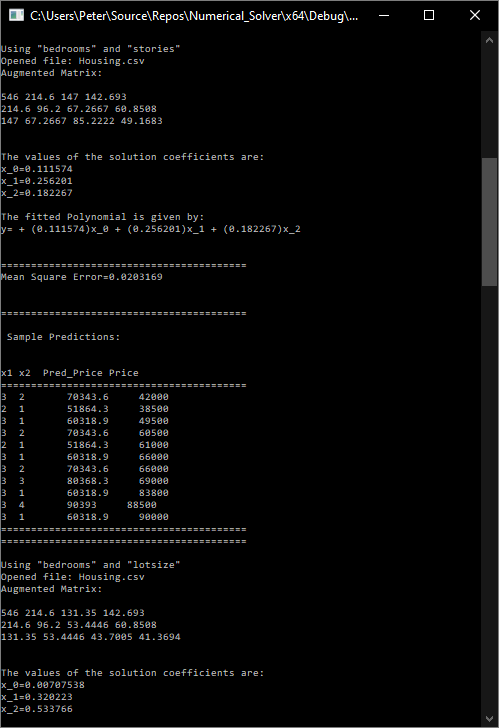
* All columns are in different scales, normalizing was used according to the following simple form: normalized\_value = (original\_value – min\_value) / (max\_value – min\_value)
* Mean Square Error was calculated based on normalized values
* It was noticed that MSE is the lowest when using "**bathrooms**" and "**lotsize**" features as per the following summary table

|  |  |
| --- | --- |
| **Fearures** | **MSE** |
| Using "bedrooms" and "bathrooms" | 0.01825 |
| Using "bedrooms" and "stories" | 0.020317 |
| Using "bedrooms" and "lotsize" | 0.016463 |
| Using "bathrooms" and "stories" | 0.017282 |
| Using "bathrooms" and "lotsize" | 0.014007 |
| Using "stories" and "lotsize" | 0.014909 |

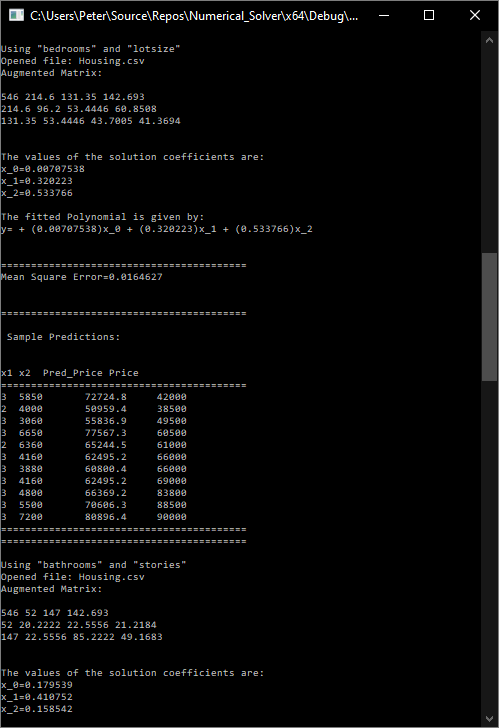
### **Using Bedrooms & Bathrooms**



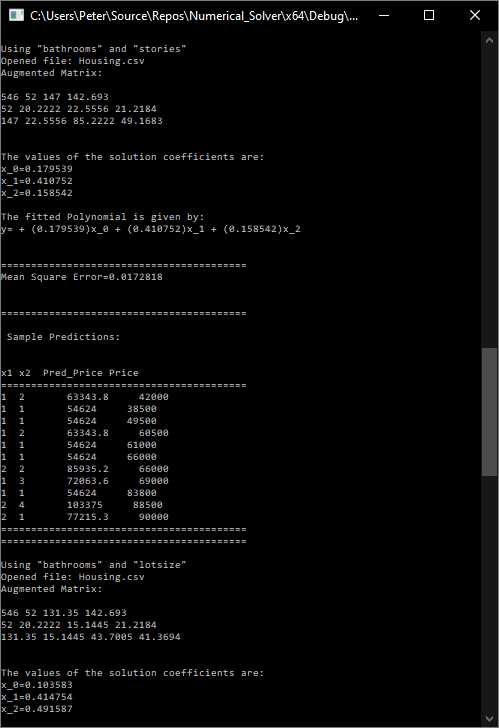
### **Using Bedrooms & Stories**



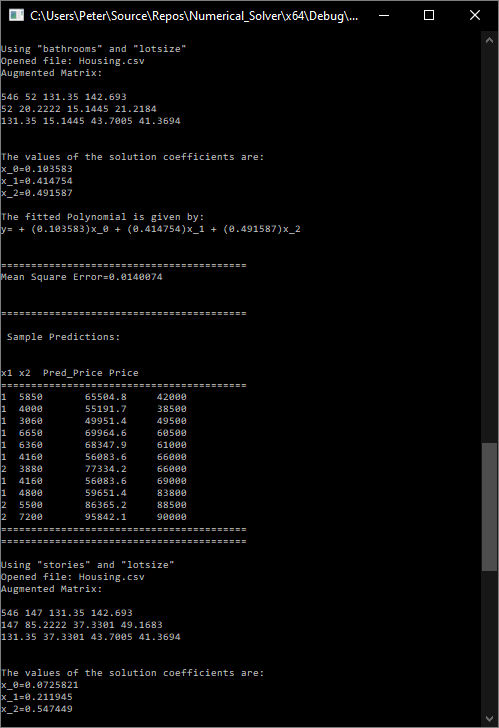
### **Using Bedrooms & Lotsize**



### **Using Bathrooms & Stories**



### **Using Bathrooms & Lotsize**



### **Using Stories & Lotsize**

