Compulsory 2 Report

Sebastian Zapart

February 19, 2024

Contents

1 Introduction

This report is my compulsory two assignment for Math 3 course. It encompasses 2 sub-assignments split into the mathematical and programming / visualization parts. For these assignments I have used C++ with OpenGL and Eigen APIs. The source code can be found by clicking: "HERE" at github.com or by using this URL: https://github.com/Shrimpy02/SebastianZapart_Math3_Compulsory2.git.

2 Assignment One

In the first assignment or 4.6.11, we are tasked in creating a parabolic curve using the method of least squares that most accurately fits description of 7-8 self-chosen points on a 2D plane. Then use C++ to program the math logic and OpenGL to visualize the function curve.

2.1 Math

The method of least squares takes M points and converts it to, in this case, an $M \times 3$ matrix. A parabolic curve is defined as a second-degree function. Since we already have the x values from our points, we define the matrix elements as $\{x^2, x, 1\}$, given by the general function $f(x) = ax^2 + bx + c$.

With the addition of an $M \times 1$ matrix containing all the y values, we have everything we need to calculate the coefficient values a, b, and c for our parabolic curve.

X	**Y**
1	1
2	-0.5
3	-1
4	-2
5	-1
6	0.5
8	1

Table 1: Table of 7 points

$$
\begin{aligned}
f(x) &= a \cdot x^2 + b \cdot x + c \cdot 1 \\
1 &= a \cdot 1^2 + b \cdot 1 + c \cdot 1 \\
-0.5 &= a \cdot 2^2 + b \cdot 2 + c \cdot 1 \\
-1 &= a \cdot 3^2 + b \cdot 3 + c \cdot 1 \\
-2 &= a \cdot 4^2 + b \cdot 4 + c \cdot 1 \\
-1 &= a \cdot 5^2 + b \cdot 5 + c \cdot 1 \\
0.5 &= a \cdot 6^2 + b \cdot 6 + c \cdot 1 \\
1 &= a \cdot 8^2 + b \cdot 8 + c \cdot 1
\end{aligned}
$$

These functions give our $7 \times 3 = A$ and $7 \times 1 = Y$, matrix.

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 4 & 2 & 1 \\ 9 & 3 & 1 \\ 16 & 4 & 1 \\ 25 & 5 & 1 \\ 36 & 6 & 1 \\ 64 & 8 & 1 \end{bmatrix} \qquad Y = \begin{bmatrix} 1 \\ -0.5 \\ -1 \\ -2 \\ -1 \\ -0.5 \\ 1 \end{bmatrix}$$

We transpose matrix A to obtain A^T. We can then multiply $A^T \cdot A$ to get B.

$$AT = \begin{bmatrix} 1 & 4 & 9 & 16 & 25 & 36 & 64 \\ 1 & 2 & 3 & 4 & 5 & 6 & 8 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \qquad A = \begin{bmatrix} 1 & 1 & 1 \\ 4 & 2 & 1 \\ 9 & 3 & 1 \\ 16 & 4 & 1 \\ 25 & 5 & 1 \\ 36 & 6 & 1 \\ 64 & 8 & 1 \end{bmatrix}$$

$$AT \cdot A = B = \begin{bmatrix} 6396 & 958 & 156 \\ 958 & 155 & 29 \\ 156 & 29 & 7 \end{bmatrix}$$

We can also find C with $A^T \cdot Y$:

$$AT = \begin{bmatrix} 1 & 4 & 9 & 16 & 25 & 36 & 64 \\ 1 & 2 & 3 & 4 & 5 & 6 & 8 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \qquad Y = \begin{bmatrix} 1 \\ -0.5 \\ -1 \\ -2 \\ -1 \\ -0.5 \\ 1 \end{bmatrix}$$

$$AT \cdot Y = C = \begin{bmatrix} 15 \\ -5 \\ -2 \end{bmatrix}$$

Then by calculating $B^{-1} \cdot C$ we get the coefficients X.

$$B^{-1} = \begin{bmatrix} 0.006 & -0.055 & 0.091 \\ -0.055 & 0.521 & -0.932 \\ 0.091 & -0.932 & 1.985 \end{bmatrix} \qquad C = \begin{bmatrix} 15 \\ -5 \\ -2 \end{bmatrix}$$

$$B^{-1} \cdot C = X = \begin{bmatrix} a = 0.186 \\ b = -1.569 \\ c = 2.085 \end{bmatrix}$$

With these coefficients we get a parabolic curve depicted by the 7 points we choose ourselves. Examples of this function from GeoGebra can be seen in figure 1 below.

$$f(x) = 0.186x^2 - 1.569x + 2.085 \tag{1}$$

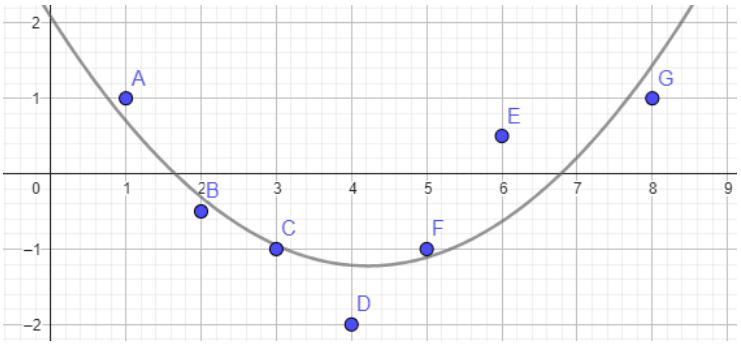

Figure 1: Function f(x) visualized with GeoGebra.

C++ Code For coding the math in these assignments as mentioned I use the Eigen API. It is a good library and handles matrix calculation and manipulation consistently and efficiently. The actual math is the same as explained above. Here you can see how I wrote it in code.

Listing 1: Finding coefficients and writing vertices to file.

```
void Main()
{
// points
const vector<Vector2f> Points = { Vector2f(1,1),
                                  Vector2f(2,-0.5),
                                  Vector2f(3,-1),
                                  Vector2f(4,-2),
                                  Vector2f(5,-1),
                                  Vector2f(6,0.5),
                                  Vector2f(8,1) };

// Maths
MatrixXf A = AOne->GenMatrixFrom7xPoints(Points1);
MatrixXf TransA = A1transpose();
MatrixXf Y = AOne->GenMatrixFrom7yPoints(Points1);
MatrixXf B = TransA * A;
MatrixXf NegB = B.inverse();
MatrixXf C = TransA * Y;
MatrixXf X = NegB*C;
AOne->GenerateVertices(X,true);
AOne->WriteVerticesToFile(6, "Assignment1VertexFile.txt");
}
```

2.2 Opengl

The `GenerateVertices` and `WriteVerticesToFile` functions create and export vertices to file based on the function given by the matrix calculation we have performed. In OpenGL the program reads and initializes the geometry, before pushing it to the render loop so that we can se the result of the calculations.

Listing 2: Initialize Geometry from file.

```
void Main{

CurrentAssignment = AssignmentOne;
// ------------------------------------------------
if (CurrentAssignment == AssignmentOne)
        Graph.Initialize(
                        Graph.generateNormalGeometryFromFile(
                                "Assignment1VertexFile.txt"), 6);
else if (CurrentAssignment == AssignmentTwo)
        Graph.Initialize(
                        Graph.generateNormalGeometryFromFile(
                                "Assignment2VertexFile.txt"), 6);
// RenderLoop
}
```

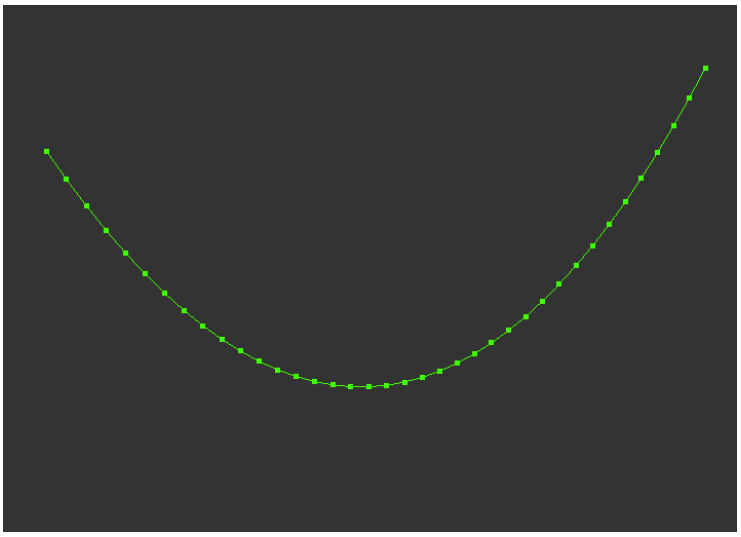

Figure 2: Function f(x) visualized with OpenGL.

3 Assignment Two

In the second assignment or 4.6.16, we are tasked in creating a function that exactly interpolates four points of our choosing on a 2D plane. Then use C++ to program the math logic and OpenGL to visualize the function curve.

3.1 Math

Since the function we are trying to interpolate must accurately pass through each point, the degree of the function must be one less than the total number of points. Therefore, when we express the x and y values, we obtain a cubic matrix.

In this case, since we have four points, we use a third-degree function represented by $f(x) = ax^3 + bx^2 + cx + d$. This choice results in a 4×4 matrix, allowing us to directly process and obtain the function coefficients without the need for additional calculations.

X	**Y**
1	5
3	-2
4	0
5	1

Table 2: Table of 4 points

$$f(x) = a * x^3 + b * x^2 + c * x + d * 1 \tag{2}$$

$$5 = a * 1^3 + b * 1^2 + c * 1 + d * 1 \tag{3}$$
$$-2 = a * 3^3 + b * 3^2 + c * 3 + d * 1 \tag{4}$$
$$0 = a * 4^3 + b * 4^2 + c * 4 + d * 1 \tag{5}$$
$$1 = a * 5^3 + b * 5^2 + c * 5 + d * 1 \tag{6}$$

These functions give our $4 \times 4 = A$ and $4 \times 1 = Y$, matrix.

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 27 & 9 & 3 & 1 \\ 64 & 16 & 4 & 1 \\ 125 & 25 & 5 & 1 \end{bmatrix} \qquad Y = \begin{bmatrix} 5 \\ -2 \\ -0 \\ 1 \end{bmatrix}$$

Since $A \cdot X = Y$, we know that $A^{-1} \cdot Y = X$.

$$A^{-1} = \begin{bmatrix} -0.04 & 0.25 & -0.3 & 0.125 \\ 0.5 & -2.5 & 3 & -1 \\ -1.95 & 7.25 & -7.16 & 2.375 \\ 2.5 & -5 & 5 & -1.5 \end{bmatrix} \qquad Y = \begin{bmatrix} 5 \\ -2 \\ -0 \\ 1 \end{bmatrix}$$

$$A^{-1} \cdot Y = X = \begin{bmatrix} -0.58 \\ 6.5 \\ -21.91 \\ 21 \end{bmatrix}$$

With these coefficients we get a third-degree function curve exactly interpolated between the 4 points we chose earlier. Example of this function from GeoGebra can be seen in figure 3 below.

$$f(x) = -0.58x^3 + 6.5x^2 - 21.91x + 21 \tag{7}$$

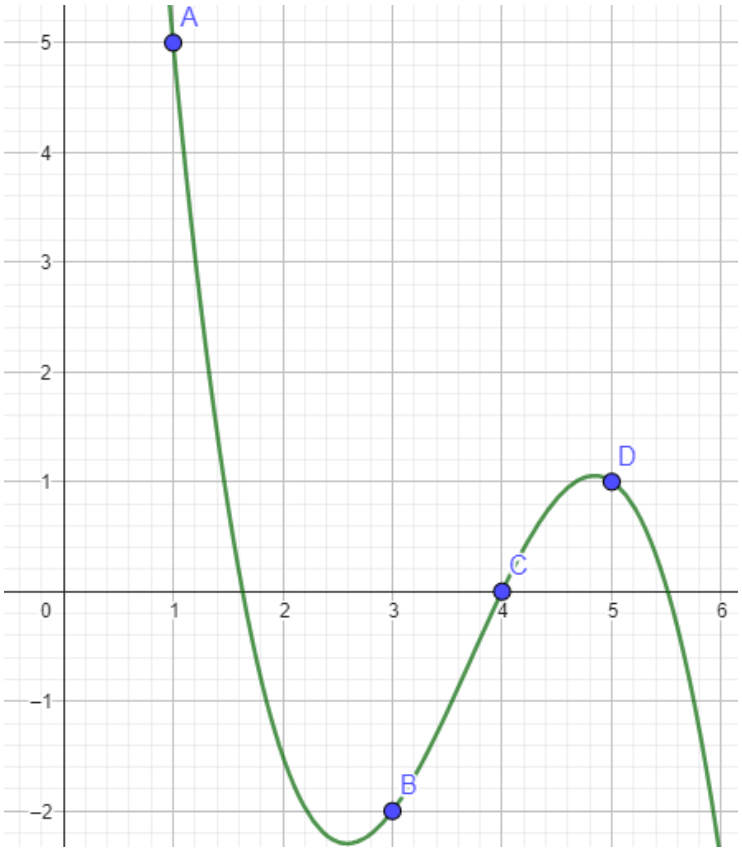

Figure 3: Function f(x) visualized with GeoGebra.

C++ Code Much like the first assignment the prosses of matrix calculations is the same as explained above.

Listing 3: Find coefficients and write to file.

```
void Main()
{
// points
const vector<Vector2f> Points = {Vector2f(1,5),
                                 Vector2f(3,-2),
                                Vector2f(4,0),
                                Vector2f(5,1) };

// Maths
MatrixXf A = ATwo->GenMatrixFrom4xPoints(Points);
MatrixXf Y = ATwo->GenMatrixFrom4yPoints(Points);
MatrixXf NegA = A2.inverse();
MatrixXf X = NegA * Y;
ATwo->GenerateVertices(X);
ATwo->WriteVerticesToFile(6, "Assignment2VertexFile.txt");
}
```

3.2 Opengl

Here we see how OpenGL's render loop processes the imported and initialized geometry. It receives the shader it uses to draw with, the location offset and the scale.

Listing 4: Draw vertecies from file

```
// Render loop
while (!glfwWindowShouldClose(window))
        {
                // ---------- Input ----------
                updateDeltaTime();

                // ProcessInput
                processInput(window);

                // ---------- Rendering options ----------
                glClearColor(0.2f, 0.2f, 0.2f, 1.0f);
                glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

                // ---------- Drawing ----------

                // Rendering
                Graph.drawGraphGeometry(&normalShader,
                                        vec3(0.0, 0.0, -3.0),
                                        vec3(1, 1, 1));

                // GLFW: swap buffers and process events
                // -------------------------------------------------------
                glfwSwapBuffers(window);
                glfwPollEvents();
        }
```

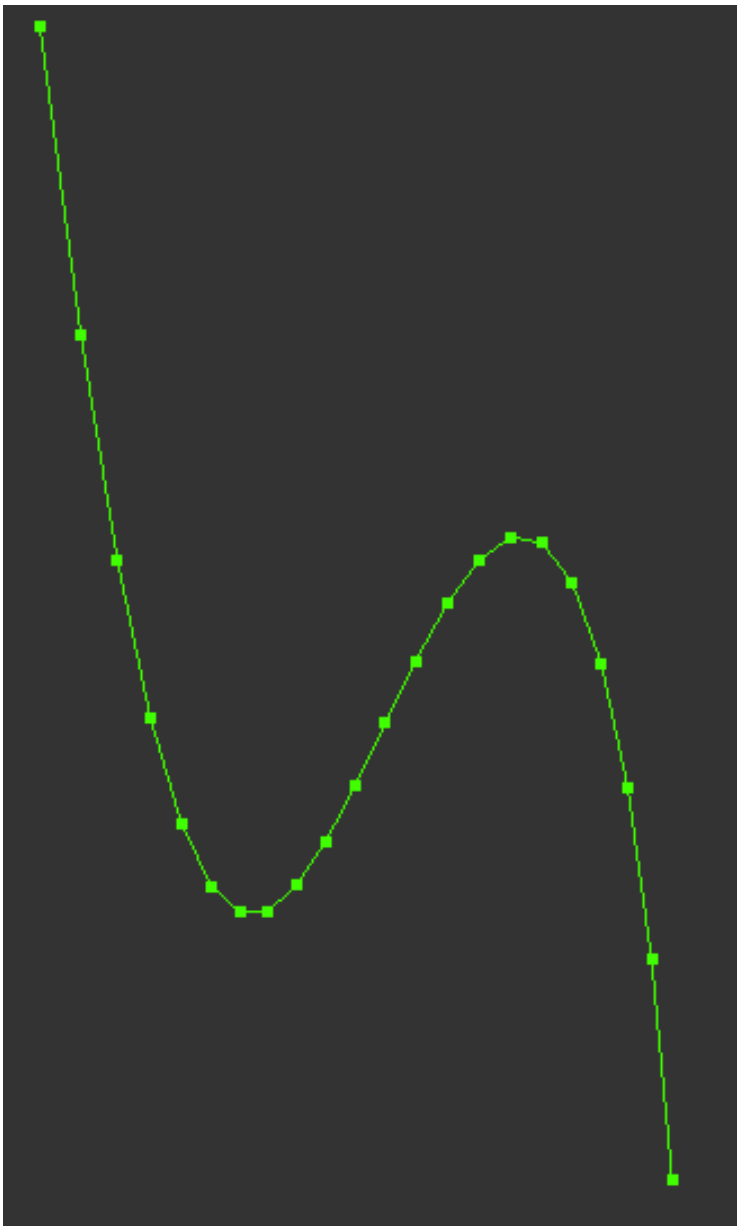

Figure 4: Function f(x) visualized with OpenGL.

4 Discussion

For this compulsory I have had a good foundation to build from because it has a lot in common with the first compulsory for this course. The only new theory is math related witch works well. Needing to learn both OpenGL and math made the first compulsory overwhelming. This compulsory in contrast made learning math easier since it was the entire focus. There is room for improvement, especially in how I coded the math to find the coefficients. It

works but does not take advantage of Eigen's flexibility. However, I am happy with my work especially since it so accurately matches the GeoGebra control models.