Project 1

Markel Pisano **markel.p@digipen.edu**
540002615

1. Description of the problem

(Wikipedia, a)

Given a set of $n+1$ data points (x_i, y_i) where no two x_i are the same, a polynomial $p : \mathbb{R} \to \mathbb{R}$ is said to *interpolate* the data if $p(x_j) = y_j$ for each $j \in \{0, 1, ..., n\}$.

2. Mathematical explanation of numerical methods

2.1 Gauss-Jordan

(Wikipedia, c)

Suppose that the interpolation polynomial is in the form

$$p(t) = a_0 + a_1 t + a_2 t^2 + ... + a_n t^n \tag{1}$$

The statement that p interpolates the data points means that

$$p(t_i) = x_i \text{ for all } i \in \{0, 1, ..., n\} \tag{2}$$

If we substitute equation (1) in here, we get a system of linear equations in the coefficients a_k. The system in matrix-vector form reads the following multiplication:

$$\begin{bmatrix} t_0^n & t_0^{n-1} & \cdots & t_0 & 1 \\ t_1^n & t_1^{n-1} & \cdots & t_1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ t_n^n & t_n^{n-1} & \cdots & t_n & 1 \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_0 \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \tag{3}$$

We have to solve this system for a_k to construct the interpolant $p(x)$. The matrix on the left is commonly referred to as a *Vandermonde matrix*.

The condition number of the *Vandermonde matrix* may be large causing large errors when computing the coefficients a_i if the system of equations is solved using **Gaussian elimination**.

2.2 Lagrange

(Wikipedia, d)

Given a set of $k+1$ data points

$$(x_0, y_0), ..., (x_j, y_j), ..., (x_k, y_k) \tag{4}$$

where no two x_j are the same, the **interpolation polynomial in the Lagrange form** is a linear combination

$$L(x) := \sum_{j=0}^{k} y_j l_j(x) \tag{5}$$

of Lagrange basis polynomials

$$l_j(x) := \prod_{\substack{0 \le m \le k \\ m \ne j}} \frac{x - x_m}{x_j - x_m} = \frac{x - x_0}{x_j - x_0} \cdots \frac{x - x_{j-1}}{x_j - x_{j-1}} \frac{x - x_{j+1}}{x_j - x_{j+1}} \cdots \frac{x - x_k}{x_j - x_k} \tag{6}$$

where $0 \le j \le k$. Note how, given the initial assumption that no two x_j are the same, then (when $m \ne j)x_j - x_m \ne 0$, so this expression is always well-defined. The reason pairs $x_i = x_j$ with $y_i \ne y_j$ are not allowed in that no interpolation function L such that $y_i = L(x_i)$ would exist; a function can only get one value for each argument x_i. On the other hand, if also $y_i = y_j$, then those two points would actually be one single point.

For all $i \ne j, l_j(x_i)$ includes the term $(x - x_i)$ in the enumerator, so the whole product will be zero at $x = x_i$:

$$\forall(j \ne i) : l_j(x_i) := \prod_{m \ne j} \frac{x_i - x_m}{x_j - x_m} = \frac{x_i - x_0}{x_j - x_0} \cdots \frac{x_i - x_i}{x_j - x_{j-1}} \cdots \frac{x_i - x_k}{x_j - x_k} = 0 \tag{7}$$

On the other hand,

$$l_j(x_j) := \prod_{m \ne j} \frac{x_j - x_m}{x_j - x_m} = 1 \tag{8}$$

In other words, all basis polynomials are zero at $x = x_j$, except $l_j(x)$, for which it holds that $l_j(x_j) = 1$, because it lacks the $(x - x_j)$ term.

It follows that $y_j l_j(x_j) = y_j$, so at each point x_j, $L(x_j) = y_i + 0 + 0 + \cdots + 0 = y_j$, showing that L interpolates the function exactly.

2.3 Newton

(Wikipedia, e)

Given a set of $k+1$ data points

$$(x_0, y_0), \cdots, (x_j, y_j), \cdots, (x_k, y_k) \tag{9}$$

where no two x_j are the same, the Newton interpolation polynomial is a linear combination of **Newton basis polynomials**

$$N(x) := \sum_{j=0}^{k} a_j n_j(x) \tag{10}$$

with the Newton basis polynomials defined as

$$n_j(x) := \prod_{i=0}^{j-1} (x - x_i) \tag{11}$$

for $j > 0$ and $n_0(x) \equiv 1$.

The coefficients are defined as

$$a_j := f[y_0, \cdots, y_j] \tag{12}$$

where $f[y_0, \cdots, y_j]$ is the notation for the divided differences.

Thus the Newton polynomial can be written as

$$N(x) = f[y_0] + f[y_0, y_1](x - x_0) + \cdots + f[y_0, \cdots, y_k](x - x_0)(x - x_1) \cdots (x - x_{k-1}) \tag{13}$$

Divided Differences

(Wikipedia, b)

Given $k+1$ data points

$$(x_0, y_0), \cdots, (x_k, y_k) \tag{14}$$

The **forward divided differences** are defined as:

$$f[y_v] := y_v, \quad v \in \{0, \cdots, k\} \tag{15}$$

$$f[y_v, \cdots, y_{v+j}] := \frac{[y_{v+1}, \cdots, y_{v+j}] - [y_v, \cdots, y_{v+j-1}]}{x_{v+j} - x_v}, \quad v \in \{0, \cdots, k-j\}, j \in \{1, \cdots, k\} \tag{16}$$

To make the recursive process more clear, the divided differences can be put in a tabular form:

$$\begin{array}{cccccc}
Node & P_0 & P_1 & P_2 & \cdots & P_k \\
 & & & & & \\
x_0 & y_0 = f[y_0] & & & & \\
 & & f[y_0, y_1] & & & \\
x_1 & y_1 = f[y_1] & & f[y_0, y_1, y_2] & & \\
 & & f[y_1, y_2] & & \ddots & \\
x_2 & y_2 = f[y_2] & & \cdots & & f[y_0, \cdots, y_k] \\
\vdots & \vdots & \cdots & & & \\
x_k & y_k = f[y_k] & & & &
\end{array} \tag{17}$$

3. Code implementation

The implemented solution is called from *interpolation.m* and takes a file single argument to read input from. It is called in the following way:

$$interpolation('input\_data') \tag{18}$$

3.1 Gauss-Jordan

For each axis computes $k+1$ coefficients for a polynomial in standard basis. Coefficient computation is done by extracting the last column in the *RREF* resultant matrix. The matrix is composed of $k+1$ rows of $v(t_i) = [1, t, t^2, \cdots, t^k]$, where t_i is the value from the **mesh selected**.

3.2 Lagrange

For each axis computes every node in the grid by applying $O(n^2)$ complexity algorithm to compute the Lagrange basis.

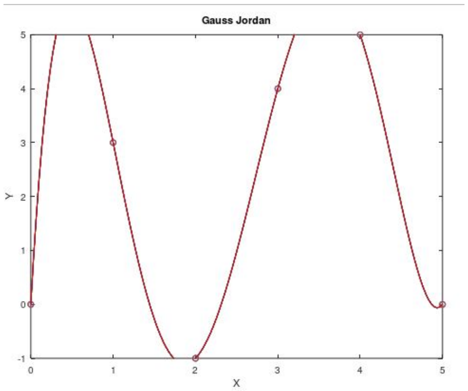
3.3 Newton

For each axis computes every node in the grid with Newton method. Due to the increasing computational cost of a recursive approach at **divide difference** computation, a dynamic solution is used instead.

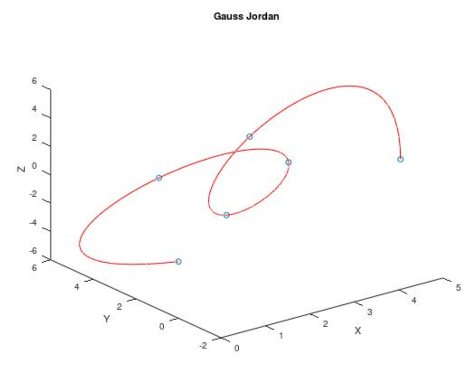
Using a triangular structure, where each level's element count is $Lvl(i) = N - i$, where $i \in \{0, \cdots, N-1\}$ and N is input point count, each level's data is filled bottom up.

4. Examples

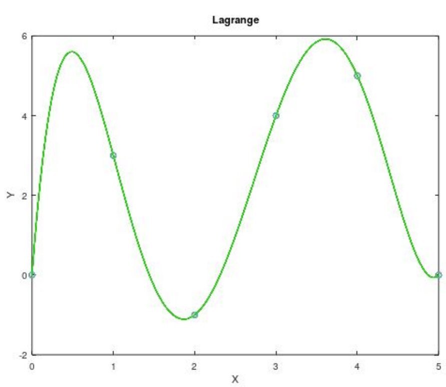
6 Points



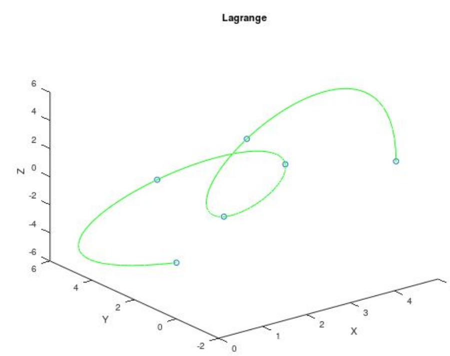
(i) time = 0.0323131s



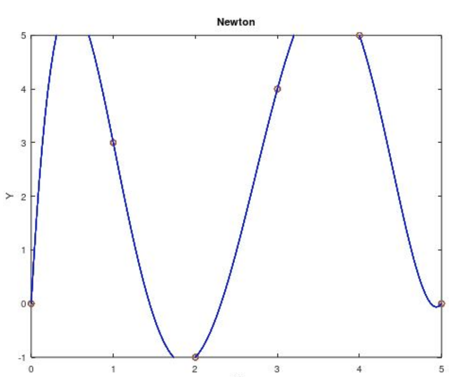
(ii) time = 0.045459s



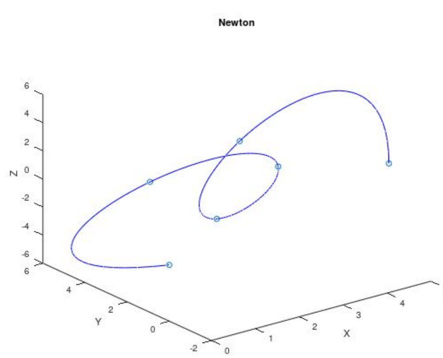
(i) time = 0.13328s



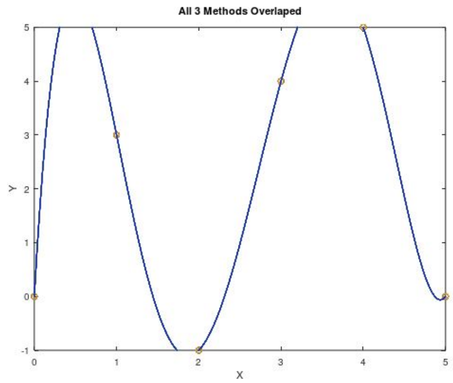
(ii) time = 0.197903s

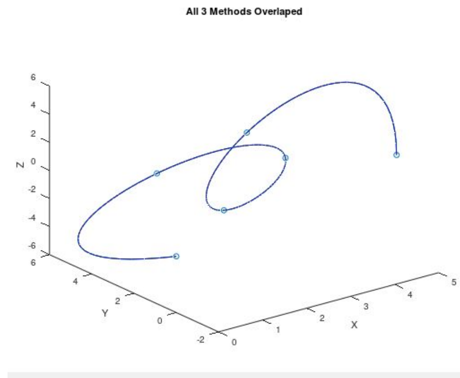


(i) time = 0.049901s

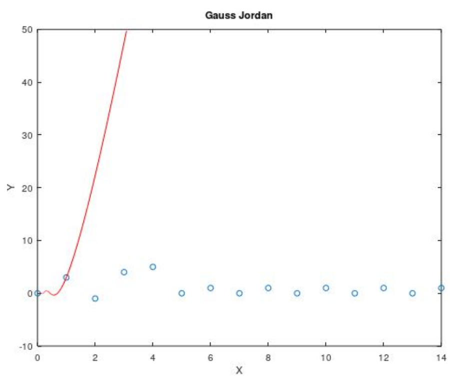


(ii) time = 0.0738909s

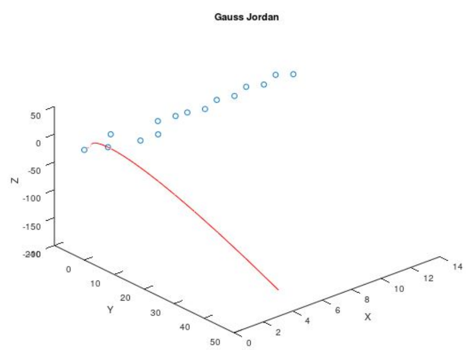




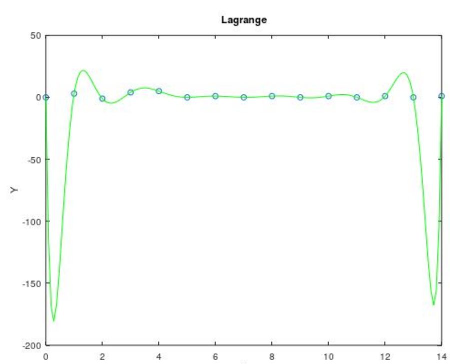
15 points



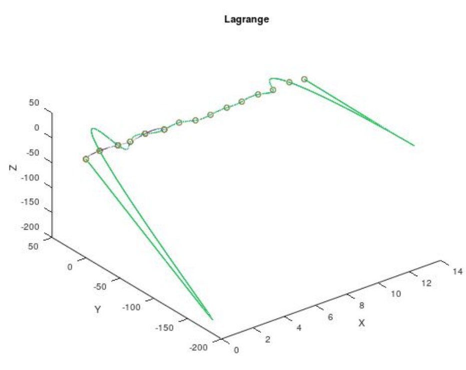
(i) time = 0.043365s



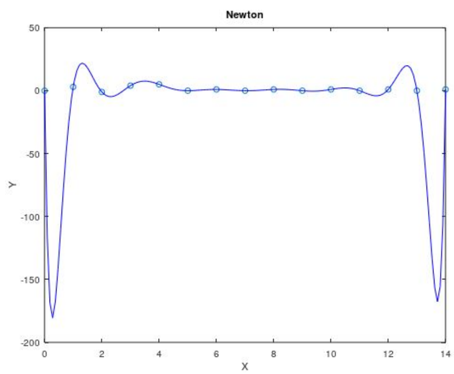
(ii) time = 0.0668411s



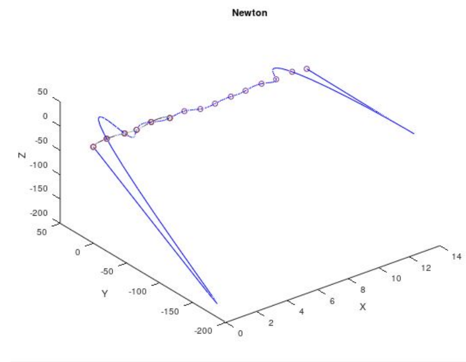
(i) time = 0.844089s



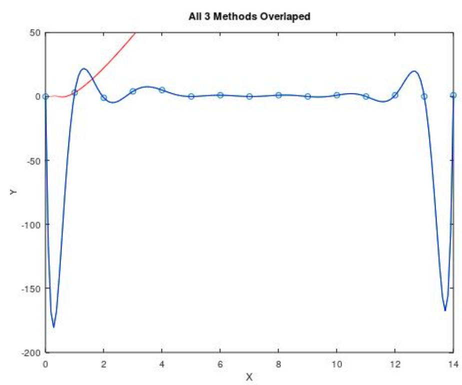
(ii) time = 1.30471s

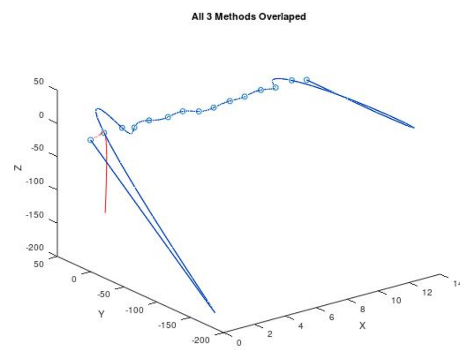


(i) time = 0.254219s



(ii) time = 0.3769s





5. Observations

By observing the example images it can be conclude that Gauss-Jordan method, although being the fastest at computation time, perform poorly with polynomials of high degree.

For a 6^{th} degree curve each 3 methods have a precise result, but as degrees are added to the polynomial, Gauss-Jordan loses lot of precision, while Lagrange and Newton don't.

References

Wikipedia. Polynomial interpolation. https://en.wikipedia.org/wiki/Polynomial_interpolation#Definition, a.

Wikipedia. Constructing the interpolation polynomial. https://en.wikipedia.org/wiki/Divided_differences, b.

Wikipedia. Constructing the interpolation polynomial. https://en.wikipedia.org/wiki/Polynomial_interpolation#Constructing_the_interpolation_polynomial, c.

Wikipedia. Constructing the interpolation polynomial. https://en.wikipedia.org/wiki/Lagrange_polynomial#Definition, d.

Wikipedia. Constructing the interpolation polynomial. https://en.wikipedia.org/wiki/Newton_polynomial#Definition, e.