REPUBLIC OF TURKEY

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

MACHINE LEARNING FOR POLYNOMIALS

SARE NUR AYDIN - SERDAR GENÇ

BACHELOR'S THESIS

COMPUTER ENGINEERING

GEBZE
2024

REPUBLIC OF TURKEY

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

MACHINE LEARNING FOR POLYNOMIALS

SARE NUR AYDIN - SERDAR GENÇ

BACHELOR’S THESIS

COMPUTER ENGINEERING

SUPERVISOR
ASST. PROF. TÜLAY AYYILDIZ

GEBZE
2024

BACHELOR’S THESIS JURY APPROVAL FORM

The thesis of Sare Nur AYDIN - Serdar GENÇ, which was defended on 16/01/2025 in front of the jury formed by the decision of the board of Gebze Technical University.

JURY

Member (Supervisor)	:	Asst. Prof. Tülay AYYILDIZ
Member	:	Prof. Dr. Didem GÖZÜPEK

APPROVAL

The decision of the board of Gebze Technical University, dated 16/01/2025.

ABSTRACT

The subject of this research is to produce polynomials with certain properties using machine learning techniques. In the study, it was resolved how to accurately produce polynomials that meet the specified conditions with machine learning methods and how artificial intelligence can be used to analyze the mathematical structures of these polynomials. The use of artificial intelligence models for this purpose was evaluated as an original and innovative approach. The research made a significant contribution to understanding the effectiveness of artificial intelligence techniques in solving mathematical problems.

As a result of the research conducted, only a few studies on polynomials conducted in the last few years were identified in the literature; some of them are given in references [1, 2, 3]. Within the scope of the study, it was predicted that when polynomials with certain properties were produced, the generated data sets could constitute an important resource in other studies on machine learning and polynomials.

In this project, it was examined how GAN (Generative Adversarial Network) [4,5] and VAE (Variational Autoencoder) [6,7] deep learning models can be used in polynomial production, the first feature determined by using GAN and VAE, ”the polynomial being positive in all cases”, was studied and the project was completed by creating GAN and VAE models that can produce positive polynomials.

ÖZET

Bu araştırmanın konusu, makine öğrenimi tekniklerini kullanarak belirli özelliklere sahip polinomlar üretmektir. Çalışmada, belirlenen koşulları sağlayan polinomların makine öğrenmesi yöntemleriyle isabetli bir şekilde üretilmesi ve bu polinomların matematiksel yapılarını analiz etmede yapay zekanın nasıl kullanılabileceği çözümlenmiştir. Yapay zeka modellerinin bu amaç doğrultusunda kullanılması özgün ve yenilikçi bir yaklaşım olarak değerlendirilmiştir. Araştırma, yapay zeka tekniklerinin matematiksel problemleri çözmedeki etkinliğini anlamak açısından önemli bir katkı sağlamıştır.

Yapılan araştırmalar sonucunda, literatürde polinomlarla ilgili yalnızca son birkaç yılda yapılmış az sayıda çalışma tespit edilmiştir; bunlardan bazıları [1, 2, 3] referanslarda verilmiştir. Çalışma kapsamında, belirli özelliklere sahip polinomlar üretildiğinde, oluşturulan veri setlerinin makine öğrenmesi ve polinomlar ile ilgili diğer çalışmalarda önemli bir kaynak oluşturabileceği öngörülmüştür.

Bu projede, GAN (Generative Adversarial Network) [4,5] ve VAE (Variational Autoencoder) [6,7] derin öğrenme modellerinin polinom üretiminde nasıl kullanılabileceği incelenmiş, GAN VE VAE kullanılarak belirlenen ilk özellik olan “polinomun her durumda pozitif olması” konusu üzerine çalışılmış ve pozitif polinom üretebilen GAN ve VAE modelleri oluşturularak proje tamamlanmıştır.

ACKNOWLEDGEMENT

We would like to thank our valuable advisor Asst. Prof. Tülay Ayyıldız, who always supported us by giving her valuable time from the beginning to the end of the ”Machine Learning for Polynomials” project, which we started with great enthusiasm due to both our interest in machine learning and our love of mathematics, and who gave us insightful ideas on how to solve every problem we encountered, and who never failed to provide feedback until the very end of the project, and Res. Asst. Başak Karakaş, who guided us especially in the initial stages of the project with her experience on GAN.

CONTENTS

1. DEFINITION OF THE PROJECT

The main goal of the ”Generating Positive Polynomials with Machine Learning” project is to produce polynomials that have positive values for all real numbers using machine learning models. In this project, Generative Adversarial Network (GAN) and Variational Autoencoder (VAE) models were trained, and their performances were compared to determine the most efficient model.

This study has shown how effective artificial intelligence techniques can be in solving mathematical problems. The positive polynomials generated have made meaningful contributions to the literature and created a valuable dataset for future studies. Additionally, the analysis of the outputs from the model demonstrated how successful deep learning models can be in generating mathematical structures.

2. MATHEMATICAL FOUNDATIONS OF POLYNOMIALS

Polynomials are extensively used in both theoretical mathematics and applied sciences. They play a critical role in mathematical modeling, numerical analysis, optimization, differential equations, and statistical estimation. In particular, they are widely employed in disciplines such as control theory, signal processing, computer graphics, and cryptography for modeling, analyzing, and solving system-related problems [13]. Moreover, in applied sciences like engineering and economics, polynomials are also utilized for solving complex problems [13].

Mathematical Definition of Polynomials[14]:

Recall: Let $p(x)$ be a univariate polynomial

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \in K[x]$$

where:

- Each a_i is a coefficient in K,
- x is a variable,
- n is a non-negative integer (the exponent),
- the coefficient a_n in the polynomial $p(x)$ is called the **leading coefficient** when n is the highest exponent,
- a_0 is the constant.

Note: The leading coefficient cannot be zero.

Now let $p(x) = p(x_1, x_2, \ldots, x_n)$ be a multivariate polynomial

$$p(x) = \sum_{d} a_d x^d \in K[x_1, x_2, \ldots, x_n]$$

where a_d is the coefficient of x^d, which is not a single variable but a monomial.

A monomial is defined as:

$$x^d = x_1^{d_1} x_2^{d_2} \ldots x_n^{d_n}, \quad d = (d_1, \ldots, d_n)$$

For example, x^3 and y^2 are both monomials. Additionally, the expression $x_1^2 x_2$ is also a monomial because:

- It consists of the variables x_1 and x_2,
- x_1 is raised to the exponent 2 (a non-negative integer),
- x_2 is implicitly raised to the exponent 1 (also a non-negative integer).

  Thus, $x_1^2 x_2$ qualifies as a monomial.

2.1. Fundamental Properties of Polynomials

2.1.1. Degree

The degree of a polynomial determines the highest exponent and directly affects the general behavior of the polynomial. It also influences how many times the curve can change direction and the complexity of its graph. While polynomials of odd degree extend to opposite directions at infinity, even-degree polynomials extend in the same direction. This behavior is crucial for analyzing the asymptotic properties of polynomials [9].

2.1.2. Coefficients

Coefficients play a fundamental role in determining the behavior of a polynomial. In particular, the sign of the leading coefficient directly affects the polynomial's behavior at the extremes. Even-degree polynomials with a positive leading coefficient approach positive values as $x \to +\infty$ and $x \to -\infty$, whereas those with a negative leading coefficient approach negative values. This relationship is of critical importance in understanding the growth rate and oscillatory tendencies of polynomials [10].

2.1.3. Roots

Polynomial roots represent the points where the function equals zero, indicating where the polynomial's graph intersects the axes. The distribution and multiplicity of the roots affect the overall shape and oscillations of the polynomial. In the project's development phase, the Sturm sequence method was used as an efficient algorithm for determining the number of real roots of polynomials. By analyzing sign changes, this method calculates the exact count of real roots [11].

2.1.4. Positivity and Negativity

Whether a polynomial remains continuously positive or negative over the entire set of real numbers is determined via derivative analysis and critical point evaluation.

Specifically, even-degree polynomials with a positive leading coefficient approach positive values as $x \to +\infty$ and $x \to -\infty$. Studies under Hilbert's 17th Problem have shown that positive polynomials can be expressed as sums of squares [12]. In this context, methods such as the Sturm sequence and derivative analysis can be used to obtain definitive results regarding the polynomial's behavior over a given interval.

3. PROJECT OBJECTIVE AND METHOD

3.1. Project Objective

The objective of this project was to develop an application, using machine learning techniques, that generates polynomials which take positive values for all real numbers. Toward this goal, Generative Adversarial Network (GAN) and Variational Autoencoder (VAE) models were successfully trained, and their performance was compared to determine the most efficient model. Within the scope of the project:

- A comprehensive dataset of polynomials with positive properties was constructed,
- GAN and VAE models were optimized and trained to produce positive polynomials,
- The performance of these models was evaluated, and the most suitable model was selected.

This study demonstrated the effective use of artificial intelligence techniques for solving mathematical problems. The resulting positive polynomials provide meaningful contributions to the literature and constitute a significant resource for future research.

3.2. Project Method

In this project, machine learning models were used to generate polynomials that yield positive values for all real numbers. Specifically, Generative Adversarial Network (GAN) and Variational Autoencoder (VAE) models were trained and compared.

Within the project:

- **Data Preparation:** A dataset with suitable properties for generating positive polynomials was created.
- **Model Training:** GAN and VAE models were optimized and trained, subsequently producing polynomials.
- **Performance Comparison:** The models were compared based on metrics such as generative capacity and accuracy.

During training, the Python programming language was used, drawing on TensorFlow and PyTorch libraries. NumPy, Pandas, and Matplotlib were chosen for data analysis and visualization.

As a result, once training was completed, the models were tested to identify the most efficient one, successfully achieving the goal of generating positive polynomials. Briefly, the models can be described as follows:

Variational Autoencoder (VAE)

- VAE is a probabilistic model that represents and reconstructs data in a low-dimensional latent space. In polynomial generation, it provides controllable and stable results.[7]

Generative Adversarial Network (GAN)

- A GAN model consists of two neural networks: a generator and a discriminator. The generator produces “fake” polynomials, while the discriminator attempts to distinguish whether those polynomials are real or fake.[5]

4. DATA SET CONSTRUCTION STAGES

The data set created in this project was specifically designed to produce positive polynomials. Certain mathematical properties were taken into account when constructing the data set, which in turn helped enhance the model's ability to generate positive polynomials.

4.1. Polynomial Coefficients

The coefficients of each polynomial determine its general shape and behavior. Careful selection of these coefficients enables the model to better learn the polynomial's tendencies.[9,10]

4.2. Polynomial Degree

The highest-degree term of a polynomial directly affects its complexity and root structure. High-degree polynomials exhibit more roots and more complex behavior. This property allows the model to learn polynomials of different degrees.[9,10]

4.3. Sign of the Leading Coefficient

The sign of the leading (or "leading term") coefficient determines the polynomial's behavior at infinity. Positive leading coefficients support the polynomial's positive values at large magnitudes of x.[9,10]

4.4. Sum of Coefficients

The sum of all coefficients provides information about the overall inclination of the polynomial. A positive sum of coefficients facilitates a positive tendency in the polynomial.[9,10]

4.5. Label

Indicates whether the polynomial is positive. Positive polynomials are labeled as 1, while those that are not positive are labeled as 0. This feature enables the model to learn how to generate positive polynomials.

4.6. Sign Changes

Using Descartes' Rule of Signs, the number of sign changes among the coefficients was counted. Fewer sign changes may suggest a more stable polynomial, increasing the likelihood of positivity.

4.6.1. Descartes' Rule of Signs

Descartes' Rule of Signs provides an upper bound on the number of positive and negative real roots of a polynomial by analyzing the sign variations in its coefficients.[10][14]

Positive Real Roots:

- Let $P(x)$ be a polynomial of degree n expressed in standard form:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0,$$

  where $a_n \neq 0$.

- Define $V(P(x))$ as the number of sign changes between consecutive nonzero coefficients of $P(x)$.

- The number of positive real roots of $P(x)$, counted with multiplicity, is at most $V(P(x))$ and differs from $V(P(x))$ by an even non-negative integer.

Negative Real Roots:

- Consider the polynomial $P(-x)$, obtained by substituting $x = -x$ into $P(x)$.

- Let $V(P(-x))$ denote the number of sign changes in $P(-x)$.

- The number of negative real roots, counted with multiplicity, is at most $V(P(-x))$ and differs from $V(P(-x))$ by an even non-negative integer.

Example:
Given the polynomial:

$$P(x) = x^5 - 3x^4 + 2x^3 - 4x^2 + 5x - 6.$$

Step 1: Positive Real Roots

- Coefficients: $1, -3, 2, -4, 5, -6$.

- Sign changes occur at: $1 \to -3$, $-3 \to 2$, $2 \to -4$, $-4 \to 5$, $5 \to -6$.
- Therefore, $V(P(x)) = 5$.
- Consequently, the number of positive real roots is in the set $\{5, 3, 1\}$.

Step 2: Negative Real Roots

- Compute $P(-x) = -x^5 - 3x^4 - 2x^3 - 4x^2 - 5x - 6$.
- Coefficients: $-1, -3, -2, -4, -5, -6$.
- Since all coefficients are negative, $V(P(-x)) = 0$.
- Thus, $P(x)$ has no negative real roots.

4.7. Number of Real Roots

The exact number of real roots of a polynomial can be determined using Sturm's Theorem, which involves constructing the Sturm sequence of the polynomial. Polynomials with no real roots intersecting the x-axis are more prone to be positive.[9][10][14]

4.7.1. Sturm Sequences

For a polynomial $P(x)$ with real coefficients, the Sturm sequence $\{P_0(x), P_1(x), \ldots, P_k(x)\}$ is defined as:

$$P_0(x) = P(x), \quad P_1(x) = P'(x), \quad P_{i+1}(x) = -\operatorname{rem}(P_{i-1}(x), P_i(x)),$$

where $P'(x)$ denotes the derivative of $P(x)$ and $\operatorname{rem}(P_{i-1}(x), P_i(x))$ is the remainder of the Euclidean division of $P_{i-1}(x)$ by $P_i(x)$.

Let $V(a)$ be the number of sign changes in the sequence evaluated at $x = a$. The exact number of distinct real roots of $P(x)$ in the interval (a, b) is:

$$V(a) - V(b).$$

Example:
Given the polynomial:

$$P(x) = x^4 + x^3 - x - 1.$$

Step 1: Construct the Sturm Sequence

$$\begin{aligned}
P_0(x) &= x^4 + x^3 - x - 1, \\
P_1(x) &= 4x^3 + 3x^2 - 1, \\
P_2(x) &= \frac{3}{16}x^2 + \frac{3}{4}x + \frac{15}{16}, \\
P_3(x) &= -32x - 64, \\
P_4(x) &= -\frac{3}{16}.
\end{aligned}$$

Step 2: Evaluate Sign Changes

- At $x = -\infty$: Sequence signs $(+, -, +, +, -)$, hence $V(-\infty) = 3$.
- At $x = +\infty$: Sequence signs $(+, +, +, -, -)$, hence $V(+\infty) = 1$.

Step 3: Compute the Number of Real Roots

$$\text{Number of real roots} = V(-\infty) - V(+\infty) = 3 - 1 = 2.$$

Therefore, the polynomial $P(x)$ has exactly 2 distinct real roots.

5. USAGE OF MACHINE LEARNING IN THE PROJECT

5.1. General Definition of Machine Learning[1

Machine learning (ML) is an artificial intelligence technique that allows computers to learn from data and improve automatically in certain tasks without human intervention. By recognizing patterns from past data and observations, and leveraging this information, machine learning aims to predict future situations, perform classifications, or generate new data similar to existing data.

At the core of machine learning lies the discovery of a mathematical relationship between input and output data. Once sufficient data is provided, the model learns this relationship and can make accurate predictions on new inputs.

5.2. Types of Machine Learning[2]

5.2.1. Supervised ML

In supervised ML, both the inputs and outputs (labels) in the dataset are taken as parameters (Labelled dataset). There are two main categories here: classification and regression. Classification algorithms are used to categorize data, while regression algorithms produce predictions.

5.2.2. Unsupervised ML

Unlike supervised ML, unsupervised ML does not require the data in the dataset to be labeled. In unsupervised ML, relationships and patterns between inputs and outputs are established. Clustering and association are two main categories here. Clustering algorithms group data based on their similarity; association algorithms aim to discover rules between one or more items within the dataset, predicting future behaviors or interpreting the data accordingly.

5.2.3. Semi-Supervised ML

In semi-supervised ML, a small portion of the dataset is labeled, while a larger portion is unlabeled. The model first performs an unsupervised learning or prediction on the unlabeled portion, and the resulting labels are then used for supervised learning.

5.2.4. Reinforcement ML

In reinforcement ML, the model takes a series of actions, and the outcome is treated in the framework of rewards and penalties; the model learns by receiving feedback in the form of such signals.

5.3. Machine Learning Models Used in the Project

A comprehensive investigation was carried out to determine which model would be chosen for use in this project. Throughout this process, various machine learning models were researched in detail, especially those that are not very popular but could be more efficient in generating polynomials that remain positive over all real numbers. As a result, it was decided to focus on GAN and VAE Models. It made sense to select among the different types of these models in the GAN architecture and then decide on the most efficient model by testing them further.

5.3.1. GAN (Generative Adversarial Networks)

GAN is an artificial neural network system composed of two subsystems[5]:

- **Generator:** Randomly takes an input and generates data.
- **Discriminator:** Determines whether the generated data is real or fake.

In other words, these two networks compete against each other. Over time, the generator learns to produce data that the discriminator believes is real, and the discriminator improves at detecting the fake data produced by the generator.

Advantages: The ability to model high diversity and complex distributions (e.g., generating polynomials with high variability).

Disadvantages: Training difficulty and instability (e.g., producing similar polynomials or encountering the "mode collapse" problem).

Types of GAN:

- **Vanilla GAN**
  - Vanilla GAN is the first and simplest type of GAN. It has two parts: a generator and a discriminator.
  - The generator creates fake data; the discriminator tries to see if this data is real or fake.
  - *Advantages:* It is simple and good for basic tasks.

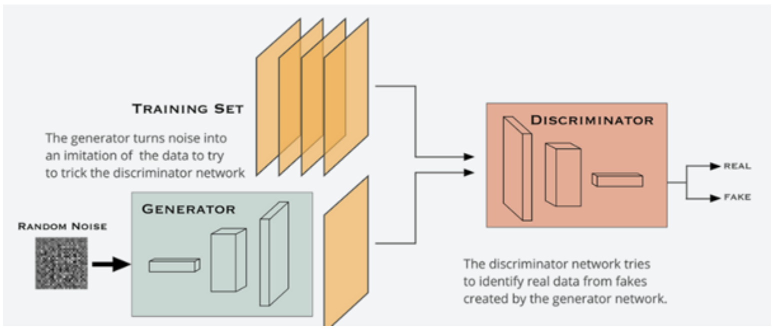

TRAINING SET
The generator turns noise into an imitation of the data to try to trick the discriminator network
RANDOM NOISE
GENERATOR
DISCRIMINATOR
REAL
FAKE
The discriminator network tries to identify real data from fakes created by the generator network.

Figure 5.1: Basic GAN architecture.

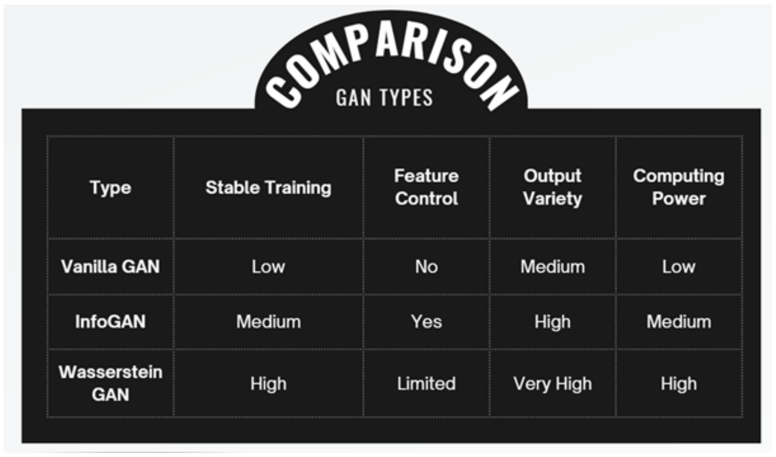
- *Disadvantages:* It can be unstable and often makes very similar outputs, called "mode collapse."
- *Best Use:* Vanilla GAN is good for straightforward data and when we don't need special control.

• **InfoGAN**

- InfoGAN is like Vanilla GAN, but it can control some features of the data.
- It uses extra information, so we can change specific things, like the number of roots in polynomials.
- *Advantages:* It helps us control features, which is useful if we want polynomials with special properties.
- *Disadvantages:* More complex and harder to train than Vanilla GAN.
- *Best Use:* InfoGAN is good when we want to control features of the data, like in polynomials.

• **Wasserstein GAN (WGAN)[4]**

- WGAN is a GAN that is better at making diverse, high-quality data.
- WGAN changes how the generator learns, so it can avoid "mode collapse."
- *Advantages:* It is stable and produces diverse results.
- *Disadvantages:* Requires more computing power and is slower than Vanilla GAN.
- *Best Use:* WGAN is best when we want stability and high variety.


COMPARISON
GAN TYPES

Type	Stable Training	Feature Control	Output Variety	Computing Power
Vanilla GAN	Low	No	Medium	Low
InfoGAN	Medium	Yes	High	Medium
Wasserstein GAN	High	Limited	Very High	High

Figure 5.2: GAN models comparison.[5]

5.3.2. VAE (Variational Autoencoder)[6,7]

VAE is a generative model composed of two main components:
Encoder:

- Takes the input data (e.g., a polynomial's coefficients).
- Converts this data into a low-dimensional latent space (vector).
- For each polynomial, the encoder outputs a mean and a variance value. These values define the distribution of the data, creating a Gaussian distribution.

Latent Space:

- Using the two parameters from the encoder, we obtain latent variables (z) from a Gaussian distribution (e.g., random sampling).
- The data is thus compressed and represented by a distribution-like state.

Decoder:

- The latent variables (z) taken from the latent space are sent to the decoder.
- The decoder transforms those latent variables back into the original data format (i.e., polynomials).

Mean represents the central point of the distribution in latent space. Most samples lie near this point. Variance determines how widely the distribution in latent space spreads. It indicates how dispersed the samples will be around the mean.

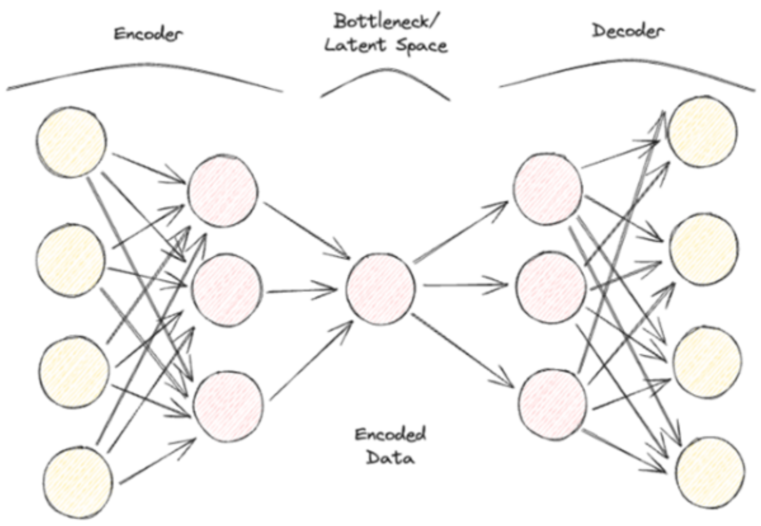


Figure 5.3: VAE structure.

VAE's Operational Logic[7]

Encoder: This component takes the input data and compresses it into a latent space.

Steps in the Encoder's Operation:

1. **Input Data Processing:** The input data x (for example, a polynomial's coefficients) is sent into the encoder neural network. It goes through various transformations in the encoder's layers.

2. **Neural Network Layers:** The encoder consists of multiple fully connected or convolutional layers (depending on data type). Each layer applies transformations to learn essential features and reduces the dimensionality.

3. **Generating Mean and Standard Deviation:**
   At the encoder's output, the network splits into two distinct heads:

   - **Mean Vector (μ):** Represents the average location for each dimension in the latent space.
   - **Log Variance Vector ($\log \sigma^2$):** Represents the spread for each dimension.

   These are computed as:

   $$\mu = W_\mu \cdot h + b_\mu$$

   $$\log \sigma^2 = W_\sigma \cdot h + b_\sigma$$

   Where:

   - W_μ and W_σ: Weight matrices for mean and variance.
   - b_μ and b_σ: Bias vectors.

- h: Final shared layer output from the encoder.

4. **Sampling from the Distribution:**
   Once μ and $\log \sigma^2$ are obtained:

   - Compute σ:
   $$\sigma = \sqrt{\exp(\log \sigma^2)}$$
   - Apply the **reparameterization trick** by sampling ϵ from a standard normal distribution $\mathcal{N}(0, 1)$:
   $$z = \mu + \sigma \cdot \epsilon$$

Latent Space and Gaussian Distribution[6]:
In a VAE, we aim for all data points in the latent space to be arranged according to this Gaussian distribution. In this setup, the data points are concentrated in the center and become sparser towards the edges. This arrangement helps the VAE to generate realistic new data samples with meaningful variations. The Gaussian distribution is given by:
$$p(z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$$

Decoder: This component takes the latent space representation and reconstructs it back to something close to the original data.

Steps in Decoder's Operation:

1. **Decoder Input:**
   A latent variable z is fed from the latent space into the decoder network.

2. **Decoder Architecture and Transformation:**
   The decoder is a neural network with multiple layers that maps from the latent space back to the original data space. Each layer in the decoder applies a transformation to move from z to a format approximating the original input.

   Each layer applies a transformation:
   $$h_{i+1} = f(W_i \cdot h_i + b_i)$$
   Where:

   - h_i: Output from the previous layer. For the first layer, $h_0 = z$.
   - W_i and b_i: Weights and biases for layer i.
   - f: Activation function introducing non-linearity (e.g., ReLU, Sigmoid, Tanh):

$$\text{ReLU: } f(x) = \max(0, x)$$
$$\text{Sigmoid: } f(x) = \frac{1}{1 + e^{-x}}$$
$$\text{Tanh: } f(x) = \tanh(x)$$

3. **Reconstruction of Original Data:**
   The final layer of the decoder outputs the reconstructed version of the input data, denoted as:
   $$\hat{x} = \text{Decoder}(z)$$

Loss Functions in VAE[7]

To effectively train the VAE, two loss functions are used:

1. Reconstruction Loss: Measures how accurately the model reconstructs the input data from the latent representation:

$$\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2$$

Here, x_i is the original data and $\hat{x}_i$ is the reconstructed data from the model.

2. KL Divergence Loss: Ensures that the latent space conforms to a Gaussian distribution:
$$D_{KL}(q(z|x)\|p(z)) = \frac{1}{2}\sum_{j=1}^{a}(\sigma_j^2 + \mu_j^2 - 1 - \log\sigma_j^2)$$

μ_j and σ_j are the mean and standard deviation for a given input. This loss guides how closely the latent space matches a true Gaussian distribution.

Total Loss: A combination of reconstruction and KL divergence losses:

$$\text{Total Loss} = \text{Reconstruction Loss} + \beta \cdot D_{KL}$$

where β is a weight controlling the balance between the two losses.

5.3.3. Comparison of VAE and GAN Models[4,5,6,7]

Criteria	**VAE (Variational Autoencoder)**	**GAN (Generative Adversarial Network)**
Training Stability	More stable (less likely to collapse)	Can be unstable, especially with complex data
Latent Space Control	Easier to control and interpret	Harder to control
Output Diversity	Limited in generating highly diverse data	Tends to generate more diverse data
Feature Control	Easier to control specific polynomial features	Possibly more unstructured except with InfoGAN
Suitability for Purpose	Good for controlled and stable polynomial generation	More suitable for complex distributions or fine-grained features
Computational Efficiency	Generally more efficient	Typically requires more computational power and time

After these comparisons, we decided to use both GAN and VAE to observe the differences. In the implementation section, we have tested both models and compared the results.

6. INFERENCES

Through experimentation and analysis, it was observed that the VAE-based model outperformed the GAN model in terms of both speed and accuracy. The VAE model consistently achieved over 90% mean training and validation accuracy, often reaching 99% - 100% in the final epochs. In contrast, the GAN model demonstrated an average accuracy of approximately 80%, with peak performance reaching 98%-99% in the later training stages.

The training time differs for model complexities, usage of different loss function methods, and other computations. In our implementations, the VAE model required less training time due to its simpler and more stable training process. The GAN model, while capable of generating more diverse polynomials according to our research, required more computational resources.[5,7]

7. SUCCESS CRITERIA

In the early stages of the project, the following success criteria were determined:

- Choosing a machine learning model that fits the purpose of the project and training a model that can generate polynomials that can be used as counterexample to disprove a theorem.
- Creating a rich dataset that can train the model well enough to allow generating complex polynomials.
- Developing an efficient and stable application that provides a user-friendly UI.

At the final stage of the project, all predetermined success criteria were successfully achieved. The GAN and VAE models were carefully selected and their implementations were successfully completed. A rich and comprehensive dataset was created to effectively support model training, ensuring the models could generate complex and accurate positive polynomials. Both models were optimized and integrated into a stable system, resulting in an efficient and robust application.

8. CHALLENGES

- During the polynomial generation process, challenges arose when dealing with polynomial degrees that lacked sufficient data for model training. To address this issue, a fallback system was implemented. In this system, the model generates a polynomial with degree n, where $n \leq d$ (the desired polynomial degree) and sufficient data exists for the model to generate it. A rule was established stating that a polynomial of degree n is considered generatable only if there are at least **25 instances** of polynomials with degree n in the dataset. This threshold ensures the generation of more diverse and accurate polynomials. To complete the polynomial up to the desired degree d, the remaining $d - n$ coefficients are randomly generated using NumPy. This process is repeated until the entire degree d polynomial satisfies the condition of being **always positive**. Each generated polynomial is stored in the database, enabling continuous improvement of the model over time.

- Another significant challenge involved determining the most effective features for the dataset. Although incorporating the determinant as a feature was initially considered, it was later excluded due to the computational complexity associated with calculating the determinant of high-degree polynomials. As an alternative, *Descartes' Rule of Signs* was utilized to account for sign changes, and *Sturm's Theorem* was employed to determine the number of real roots, enhancing the dataset with more informative features.

- Initial model training exclusively relied on polynomials that were always positive. This approach led to overfitting, causing the models to generate outputs similar to those in the training set. Additionally, the accuracy metric remained nearly constant across epochs, indicating a lack of genuine learning. To mitigate this issue, the dataset was expanded to include polynomials that were not always positive. This adjustment fostered greater model creativity and improved generalization performance.

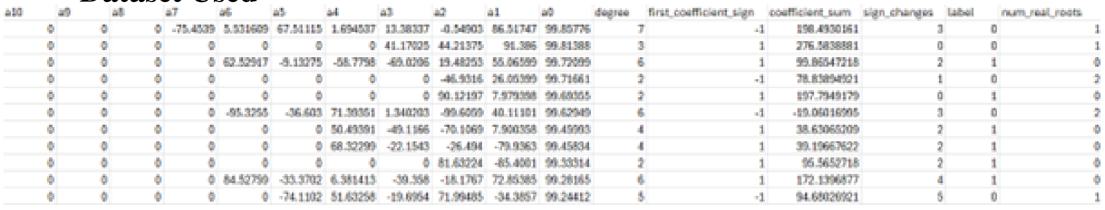
9. FUTURE IMPROVEMENTS

The related project, which generates positive polynomials using GAN and VAE machine learning models, aimed to generate positive polynomials as the first goal. In the continuation of the project, which is successful in this regard, studies can be carried out on other specific features by using appropriate machine learning models so that the user can generate more specialized polynomials. The project is open to development in this regard.

10. APPENDICES

10.1. Dataset Visuals and Outputs

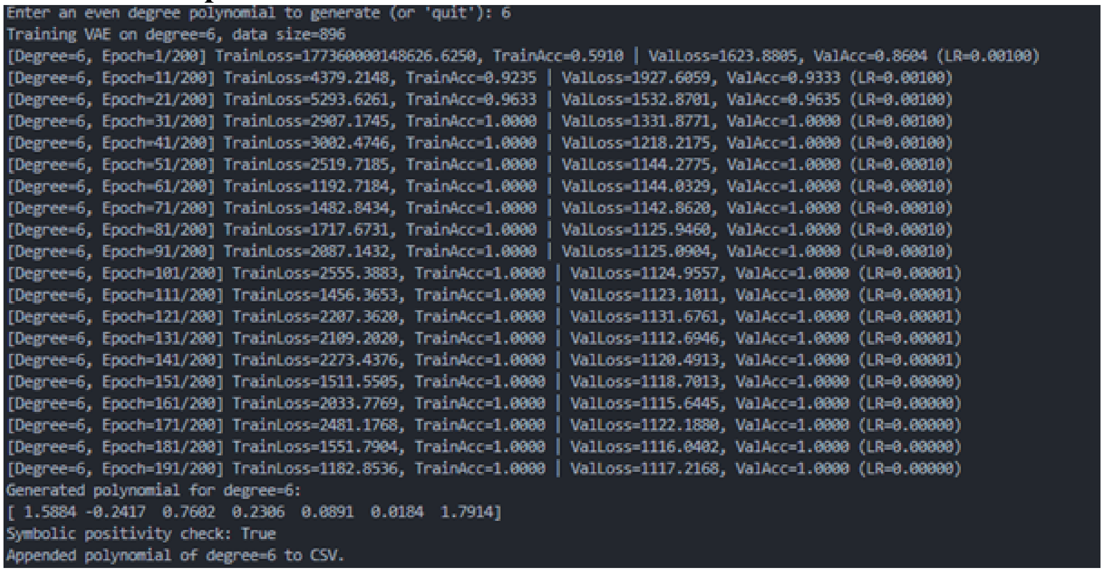
Dataset Used



a10	a9	a8	a7	a6	a5	a4	a3	a2	a1	a0	degree	first_coefficient_sign	coefficient_sum	sign_changes	label	num_real_roots
0	0	0	-75.4539	5.531609	67.51115	1.694537	13.38337	-0.54903	86.51747	99.85776	7	-1	198.4930161	3	0	1
0	0	0	0	0	0	0	41.17025	44.21375	91.386	99.81388	3	1	276.5838881	0	0	1
0	0	0	0	62.52917	-8.13275	-58.7798	-69.0206	19.48253	55.06599	99.72099	6	1	99.86547218	2	1	0
0	0	0	0	0	0	0	0	-46.9316	26.05399	99.71661	2	-1	78.83894921	1	0	2
0	0	0	0	0	0	0	0	90.12197	7.979398	99.69355	2	1	197.7949179	0	1	0
0	0	0	0	-95.3255	-36.603	71.39351	1.340203	-99.6059	40.11101	99.62949	6	-1	-19.06016995	3	0	2
0	0	0	0	0	0	50.49391	-49.1166	-70.1069	7.900358	99.45993	4	1	38.63065209	2	1	0
0	0	0	0	0	0	68.32299	-22.1543	-26.494	-79.9363	99.45834	4	1	39.19667622	2	1	0
0	0	0	0	0	0	0	0	81.63224	-85.4001	99.33314	2	1	95.5652718	2	1	0
0	0	0	0	84.52759	-33.3702	6.381413	-39.358	-18.1767	72.85385	99.28165	6	1	172.1396877	4	1	0
0	0	0	0	0	-74.1102	51.63258	-19.6954	71.99485	-34.3857	99.24412	5	-1	94.68026921	5	0	1

Figure 10.1: The dataset used for training the models.

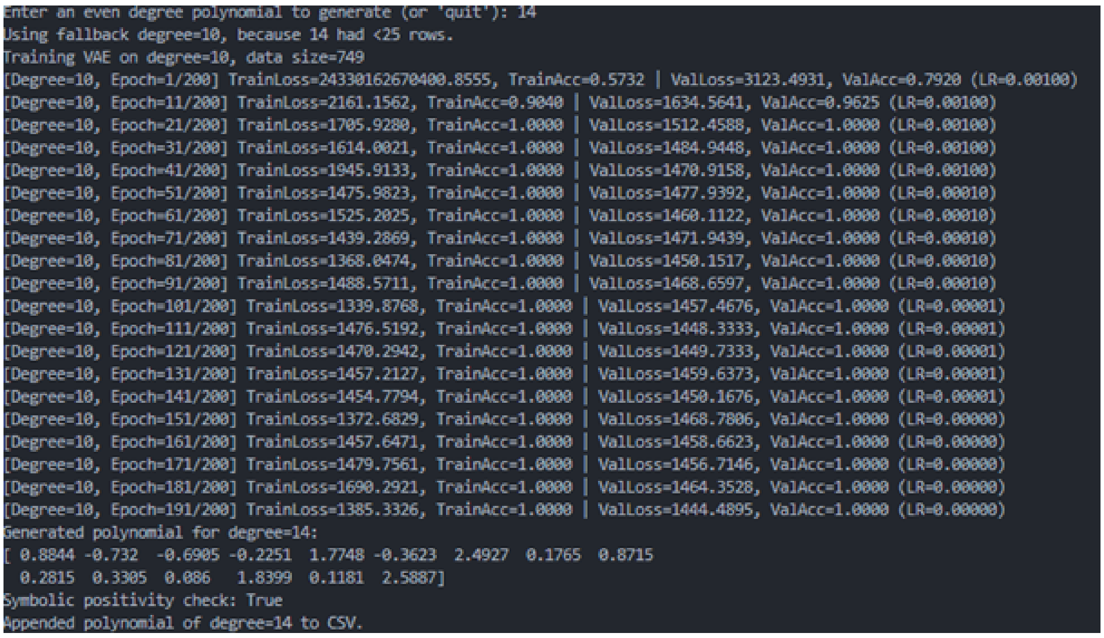
VAE Output



```
Enter an even degree polynomial to generate (or 'quit'): 6
Training VAE on degree=6, data size=896
[Degree=6, Epoch=1/200] TrainLoss=177360000148626.6250, TrainAcc=0.5910 | ValLoss=1623.8805, ValAcc=0.8604 (LR=0.00100)
[Degree=6, Epoch=11/200] TrainLoss=4379.2148, TrainAcc=0.9235 | ValLoss=1927.6059, ValAcc=0.9333 (LR=0.00100)
[Degree=6, Epoch=21/200] TrainLoss=5293.6261, TrainAcc=0.9633 | ValLoss=1532.8701, ValAcc=0.9635 (LR=0.00100)
[Degree=6, Epoch=31/200] TrainLoss=2907.1745, TrainAcc=1.0000 | ValLoss=1331.8771, ValAcc=1.0000 (LR=0.00100)
[Degree=6, Epoch=41/200] TrainLoss=3002.4746, TrainAcc=1.0000 | ValLoss=1218.2175, ValAcc=1.0000 (LR=0.00100)
[Degree=6, Epoch=51/200] TrainLoss=2519.7185, TrainAcc=1.0000 | ValLoss=1144.2775, ValAcc=1.0000 (LR=0.00010)
[Degree=6, Epoch=61/200] TrainLoss=1192.7184, TrainAcc=1.0000 | ValLoss=1144.0329, ValAcc=1.0000 (LR=0.00010)
[Degree=6, Epoch=71/200] TrainLoss=1482.8434, TrainAcc=1.0000 | ValLoss=1142.8620, ValAcc=1.0000 (LR=0.00010)
[Degree=6, Epoch=81/200] TrainLoss=1717.6731, TrainAcc=1.0000 | ValLoss=1125.9460, ValAcc=1.0000 (LR=0.00010)
[Degree=6, Epoch=91/200] TrainLoss=2887.1432, TrainAcc=1.0000 | ValLoss=1125.0904, ValAcc=1.0000 (LR=0.00010)
[Degree=6, Epoch=101/200] TrainLoss=2555.3883, TrainAcc=1.0000 | ValLoss=1124.9557, ValAcc=1.0000 (LR=0.00001)
[Degree=6, Epoch=111/200] TrainLoss=1456.3653, TrainAcc=1.0000 | ValLoss=1123.1011, ValAcc=1.0000 (LR=0.00001)
[Degree=6, Epoch=121/200] TrainLoss=2207.3620, TrainAcc=1.0000 | ValLoss=1131.6761, ValAcc=1.0000 (LR=0.00001)
[Degree=6, Epoch=131/200] TrainLoss=2109.2020, TrainAcc=1.0000 | ValLoss=1112.6946, ValAcc=1.0000 (LR=0.00001)
[Degree=6, Epoch=141/200] TrainLoss=2273.4376, TrainAcc=1.0000 | ValLoss=1120.4913, ValAcc=1.0000 (LR=0.00001)
[Degree=6, Epoch=151/200] TrainLoss=1511.5505, TrainAcc=1.0000 | ValLoss=1118.7013, ValAcc=1.0000 (LR=0.00000)
[Degree=6, Epoch=161/200] TrainLoss=2033.7769, TrainAcc=1.0000 | ValLoss=1115.6445, ValAcc=1.0000 (LR=0.00000)
[Degree=6, Epoch=171/200] TrainLoss=2481.1768, TrainAcc=1.0000 | ValLoss=1122.1880, ValAcc=1.0000 (LR=0.00000)
[Degree=6, Epoch=181/200] TrainLoss=1551.7904, TrainAcc=1.0000 | ValLoss=1116.0402, ValAcc=1.0000 (LR=0.00000)
[Degree=6, Epoch=191/200] TrainLoss=1182.8536, TrainAcc=1.0000 | ValLoss=1117.2168, ValAcc=1.0000 (LR=0.00000)
Generated polynomial for degree=6:
[ 1.5884 -0.2417  0.7602  0.2306  0.0891  0.0184  1.7914]
Symbolic positivity check: True
Appended polynomial of degree=6 to CSV.
```

Figure 10.2: Output results from the VAE model.

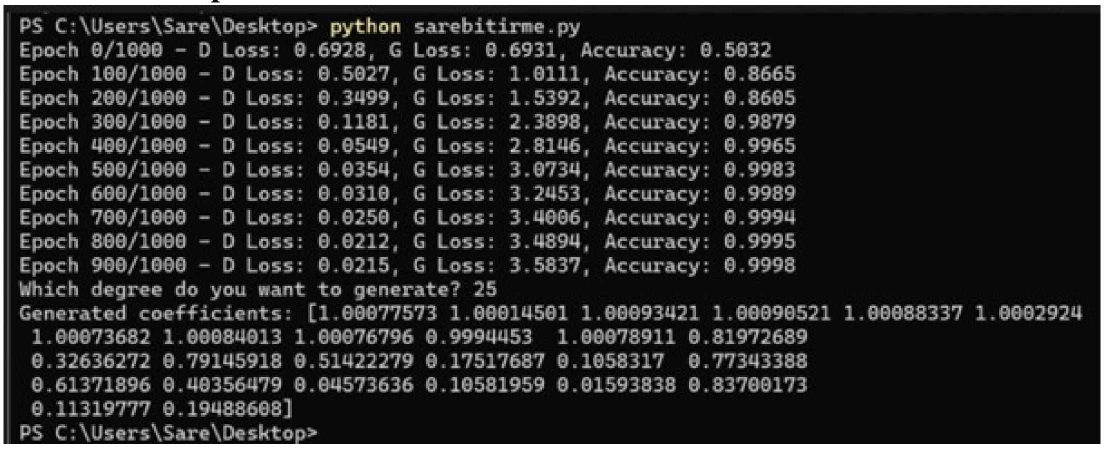
VAE Output with Fallback Degree



```
Enter an even degree polynomial to generate (or 'quit'): 14
Using fallback degree=10, because 14 had <25 rows.
Training VAE on degree=10, data size=749
[Degree=10, Epoch=1/200] TrainLoss=24330162670400.8555, TrainAcc=0.5732 | ValLoss=3123.4931, ValAcc=0.7920 (LR=0.00100)
[Degree=10, Epoch=11/200] TrainLoss=2161.1562, TrainAcc=0.9040 | ValLoss=1634.5641, ValAcc=0.9625 (LR=0.00100)
[Degree=10, Epoch=21/200] TrainLoss=1705.9280, TrainAcc=1.0000 | ValLoss=1512.4588, ValAcc=1.0000 (LR=0.00100)
[Degree=10, Epoch=31/200] TrainLoss=1614.0021, TrainAcc=1.0000 | ValLoss=1484.9448, ValAcc=1.0000 (LR=0.00100)
[Degree=10, Epoch=41/200] TrainLoss=1945.9133, TrainAcc=1.0000 | ValLoss=1470.9158, ValAcc=1.0000 (LR=0.00100)
[Degree=10, Epoch=51/200] TrainLoss=1475.9823, TrainAcc=1.0000 | ValLoss=1477.9392, ValAcc=1.0000 (LR=0.00010)
[Degree=10, Epoch=61/200] TrainLoss=1525.2025, TrainAcc=1.0000 | ValLoss=1460.1122, ValAcc=1.0000 (LR=0.00010)
[Degree=10, Epoch=71/200] TrainLoss=1439.2869, TrainAcc=1.0000 | ValLoss=1471.9439, ValAcc=1.0000 (LR=0.00010)
[Degree=10, Epoch=81/200] TrainLoss=1368.0474, TrainAcc=1.0000 | ValLoss=1450.1517, ValAcc=1.0000 (LR=0.00010)
[Degree=10, Epoch=91/200] TrainLoss=1488.5711, TrainAcc=1.0000 | ValLoss=1468.6597, ValAcc=1.0000 (LR=0.00010)
[Degree=10, Epoch=101/200] TrainLoss=1339.8768, TrainAcc=1.0000 | ValLoss=1457.4676, ValAcc=1.0000 (LR=0.00001)
[Degree=10, Epoch=111/200] TrainLoss=1476.5192, TrainAcc=1.0000 | ValLoss=1448.3333, ValAcc=1.0000 (LR=0.00001)
[Degree=10, Epoch=121/200] TrainLoss=1470.2942, TrainAcc=1.0000 | ValLoss=1449.7333, ValAcc=1.0000 (LR=0.00001)
[Degree=10, Epoch=131/200] TrainLoss=1457.2127, TrainAcc=1.0000 | ValLoss=1459.6373, ValAcc=1.0000 (LR=0.00001)
[Degree=10, Epoch=141/200] TrainLoss=1454.7794, TrainAcc=1.0000 | ValLoss=1450.1676, ValAcc=1.0000 (LR=0.00001)
[Degree=10, Epoch=151/200] TrainLoss=1372.6829, TrainAcc=1.0000 | ValLoss=1468.7806, ValAcc=1.0000 (LR=0.00000)
[Degree=10, Epoch=161/200] TrainLoss=1457.6471, TrainAcc=1.0000 | ValLoss=1458.6623, ValAcc=1.0000 (LR=0.00000)
[Degree=10, Epoch=171/200] TrainLoss=1479.7561, TrainAcc=1.0000 | ValLoss=1456.7146, ValAcc=1.0000 (LR=0.00000)
[Degree=10, Epoch=181/200] TrainLoss=1690.2921, TrainAcc=1.0000 | ValLoss=1464.3528, ValAcc=1.0000 (LR=0.00000)
[Degree=10, Epoch=191/200] TrainLoss=1385.3326, TrainAcc=1.0000 | ValLoss=1444.4895, ValAcc=1.0000 (LR=0.00000)
Generated polynomial for degree=14:
[ 0.8844 -0.732  -0.6905 -0.2251  1.7748 -0.3623  2.4927  0.1765  0.8715
  0.2815  0.3305  0.086   1.8399  0.1181  2.5887]
Symbolic positivity check: True
Appended polynomial of degree=14 to CSV.
```

Figure 10.3: Output results from the VAE model with fallback degree.

GAN Output



```
PS C:\Users\Sare\Desktop> python sarebitirme.py
Epoch 0/1000 - D Loss: 0.6928, G Loss: 0.6931, Accuracy: 0.5032
Epoch 100/1000 - D Loss: 0.5027, G Loss: 1.0111, Accuracy: 0.8665
Epoch 200/1000 - D Loss: 0.3499, G Loss: 1.5392, Accuracy: 0.8605
Epoch 300/1000 - D Loss: 0.1181, G Loss: 2.3898, Accuracy: 0.9879
Epoch 400/1000 - D Loss: 0.0549, G Loss: 2.8146, Accuracy: 0.9965
Epoch 500/1000 - D Loss: 0.0354, G Loss: 3.0734, Accuracy: 0.9983
Epoch 600/1000 - D Loss: 0.0310, G Loss: 3.2453, Accuracy: 0.9989
Epoch 700/1000 - D Loss: 0.0250, G Loss: 3.4006, Accuracy: 0.9994
Epoch 800/1000 - D Loss: 0.0212, G Loss: 3.4894, Accuracy: 0.9995
Epoch 900/1000 - D Loss: 0.0215, G Loss: 3.5837, Accuracy: 0.9998
Which degree do you want to generate? 25
Generated coefficients: [1.00077573 1.00014501 1.00093421 1.00090521 1.00088337 1.0002924
 1.00073682 1.00084013 1.00076796 0.9994453  1.00078911 0.81972689
 0.32636272 0.79145918 0.51422279 0.17517687 0.1058317  0.77343388
 0.61371896 0.40356479 0.04573636 0.10581959 0.01593838 0.83700173
 0.11319777 0.19488608]
PS C:\Users\Sare\Desktop>
```

Figure 10.4: Output results from the GAN model.

10.2. References

1. Alexandr Andoni, Rina Panigrahy, Gregory Valiant, Li Zhang, "Learning Polynomials with Neural Networks," *Proceedings of the 31st International Conference on Machine Learning*, PMLR 32(2):1908-1916, 2014.

2. Dorian Florescu, Matthew England, "Algorithmically Generating New Algebraic Features of Polynomial Systems for Machine Learning," *Proceedings of the 4th Workshop on Satisfiability Checking and Symbolic Computation (SC2 '19)*, 12 pages, CEUR Workshop Proceedings 2460, 2019.

3. Shahrzad Jamshidi, Sonja Petrović, "The Spark Randomizer: A Learned Randomized Framework for Computing Gröbner Bases," *https://arxiv.org/abs/2306.08279* (2023).

4. Martin Arjovsky, Soumith Chintala, Léon Bottou, "Wasserstein Generative Adversarial Networks," *Proceedings of the 34th International Conference on Machine Learning*, PMLR 70:214-223, 2017.

5. I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, "Generative Adversarial Networks," (2014).

6. Lucas Pinheiro Cinelli, Matheus Araújo Marins, Eduardo Antúnio Barros da Silva, Sérgio Lima Netto, "Variational Methods for Machine Learning with Applications to Deep Networks," 1st ed., 2021.

7. Carl Doersch, "Tutorial on Variational Autoencoders," *Carnegie Mellon / UC Berkeley*, August 16, 2016 (minor revisions on January 3, 2021).

8. Sebastian Raschka, Yuxi (Hayden) Liu, Vahid Mirjalili, "Machine Learning with PyTorch and Scikit-Learn," 1st ed., 2022.

9. Victor V. Prasolov, "Polynomials," *Springer*, 2004.

10. Q. I. Rahman, G. Schmeisser, "Analytic Theory of Polynomials," *Oxford University Press*, 2002.

11. Peter Borwein, Tamás Erdélyi, "Polynomials and Polynomial Inequalities," *Springer*, 1995.

12. E. Artin, "Über die Zerlegung definiter Funktionen in Quadrate," *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 5, 85–99, 1927.

13. J. W. Brown, R. V. Churchill, "Polynomials and Their Applications," In: *Applicable Mathematics*, Palgrave, London, 1990.

14. David S. Dummit, Richard M. Foote, "Abstract Algebra," *Wiley*, 3rd Edition, 2004.