

T.R.
GEBZE TECHNICAL UNIVERSITY
GRADUATE SCHOOL

MACHINE LEARNING AND SYMBOLIC COMPUTATION



BAŞAK KARAKAŞ

A THESIS OF MASTER OF SCIENCE
DEPARTMENT OF COMPUTER ENGINEERING

ADVISOR: ASSOC. PROF. DR. MEHMET GÖKTÜRK

MAY 2025

T.R.
GEBZE TECHNICAL UNIVERSITY
GRADUATE SCHOOL

**MACHINE LEARNING AND SYMBOLIC
COMPUTATION**

BAŞAK KARAKAŞ

A THESIS OF MASTER OF SCIENCE
DEPARTMENT OF COMPUTER ENGINEERING

ADVISOR: ASSOC. PROF. DR. MEHMET GÖKTÜRK

CO-ADVISOR: ASSIST. PROF. DR. ZAFEIRAKIS
ZAFEIRAKOPOULOS

MAY 2025

T.C.
GEBZE TEKNİK ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

MAKİNE ÖĞRENMESİ VE SEMBOLİK HESAPLAMA

BAŞAK KARAKAŞ

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMAN: DOÇ. DR. MEHMET GÖKTÜRK
II. DANIŞMAN: DR. OGR. UYESİ. ZAFEIRAKIS
ZAFEIRAKOPOULOS

MAYIS 2025

MASTER of SCIENCE JURY APPROVAL FORM

A thesis submitted by Başak, Karakaş, defended on 23/05/2025 before the jury formed with the 03/03/2025 date and 2025/17 numbered decision of the GTU Graduate Administration Board, has been accepted as a MASTER of SCIENCE thesis in the Department of Computer Engineering.

JURY

MEMBER

(THESIS ADVISOR) : Assoc. Prof. Dr. Mehmet GÖKTÜRK

MEMBER

: Assist. Prof. Dr. Tülay AYYILDIZ

MEMBER

: Assoc. Prof. Dr. Pınar ONAY DURDU

APPROVAL

Gebze Technical University Graduate Administration Board
...../...../..... date and/..... numbered decision.

SIGNATURE/SEAL

*To **my beloved family**,
for their unwavering support throughout this journey,
and to **my cats**,
who never stopped caring for me
— especially during the most intense final stages of this work.*



ABSTRACT

This thesis investigates the capability of deep generative models—particularly Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs)—to symbolically generate univariate polynomials with only real roots. Real-rooted polynomials are central in fields such as numerical stability, control systems, and approximation theory. However, generating such objects with explicit algebraic constraints remains an open challenge in the context of machine learning.

Based on the assumption that real-rootedness can be indirectly learned from coefficient space distributions, this work hypothesizes that generative models like GANs and VAEs can capture this structure using structured datasets. To test this, symbolic datasets were constructed in both monomial and Bernstein bases, containing rational and irrational root structures across 56 algebraic classes.

A GAN was trained to directly generate valid instances through adversarial learning, while a VAE was used to analyze latent structure and disentanglement in root-related features. The study evaluates performance across different degrees, bases, and symbolic complexities, revealing patterns that affect the learnability of real-rootedness.

The findings suggest that, with appropriately designed datasets and model architectures, generative models can indeed learn to produce real-rooted polynomials with high accuracy. This thesis contributes to the field of algebraically constrained symbolic generation and opens new avenues for integrating deep learning with algebraic reasoning.

Keywords: Real-Rooted Polynomials, GAN, Algebraic Basis, Deep Learning, Symbolic Datasets

ÖZET

Bu tez, derin üretici modellerin —özellikle Generative Adversarial Network (GAN) ve Variational Autoencoder (VAE) mimarilerinin— yalnızca reel köklere sahip tek değişkenli polinomları sembolik düzeyde üretme kapasitesini araştırmaktadır. Reel köklü polinomlar, nümerik stabilite, kontrol sistemleri ve yaklaşım kuramı gibi birçok teorik ve uygulamalı alanda temel öneme sahiptir. Ancak bu tür polinomların, hem sembolik olarak hem de açık cebirsel kısıtlarla (örneğin tüm köklerin $\mathbb{R}$ üzerinde olması) üretimi, mevcut makine öğrenmesi modelleri açısından büyük ölçüde çözülmemiş bir problemidir.

Bu çalışmada, reel köklülüğün katsayı uzayında dolaylı olarak öğrenilebileceği varsayımına dayanarak, GAN ve VAE gibi üretici modellerin bu özelliği veri üzerinden istatistiksel olarak öğrenebileceği hipotezi öne sürülmektedir. Hipotezi test etmek amacıyla, rasyonel ve irrasyonel kök yapılarına sahip polinomlardan oluşan, hem monomial hem de Bernstein bazlarında tanımlanmış özgün ve sembolik olarak yapılandırılmış veri kümeleri oluşturulmuştur. Bu veri kümeleri, her biri reel kök koşulunu sağlayan 56 farklı cebirsel yapı içermektedir.

Çalışma kapsamında, GAN modeli adversarial öğrenme aracılığıyla doğrudan reel köklü örnekler üretmek üzere eğitilmiş; VAE modeli ise latent uzaydaki yapıyı analiz etmek ve kök özelliklerine göre ayrışma (disentanglement) davranışını gözlemlemek amacıyla kullanılmıştır. Farklı derece, baz ve sembolik karmaşıklıklara göre yapılan karşılaştırmalı analizler, reel köklülüğün hangi yapısal koşullarda daha iyi öğrenilebildiğine dair niteliksel ve niceliksel bulgular sunmaktadır.

Sonuçlar, uygun biçimde yapılandırılmış veri kümeleri ve model mimarileri ile GAN ve VAE gibi üretici modellerin reel köklü polinom üretimini yüksek doğrulukla gerçekleştirebildiğini göstermektedir. Bu tez, cebirsel olarak kısıtlanmış sembolik nesne üretimi alanına katkı sağlamakta ve derin öğrenme ile cebirsel yapılar arasında daha güçlü entegrasyonlara olanak tanımaktadır.

Anahtar Kelimeler: Reel Köklü Polinomlar, GAN, Cebirsel Bazlar, Derin Öğrenme, Sembolik Veri Kümeleri

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Assoc. Prof. Dr. Mehmet Göktürk, for his constant guidance and insightful suggestions throughout this thesis work. I am deeply thankful to my co-advisor, Assist. Prof. Dr. Zefairakis Zafeirakopoulos, who supported me in every step since the very beginning of this study and consistently encouraged me with his valuable feedback. I also thank Assist. Prof. Dr. Tülay Ayyıldız for her helpful contributions and technical advice during the development of this research.



TABLE OF CONTENTS

	<u>Page</u>
ABSTRACT	vi
ÖZET	vii
ACKNOWLEDGEMENT	viii
TABLE of CONTENTS	ix
LIST of ABBREVIATIONS and ACRONYMS	xi
LIST of FIGURES	xii
LIST of TABLES	xiv
1. INTRODUCTION	1
2. RELATED WORK	3
2.1. Classical Methods for Real-rooted Polynomial Generation	3
2.2. Machine Learning in Symbolic Mathematics	4
2.3. Generative Models for Mathematical Object Synthesis	6
2.4. Polynomial Basis and Learnability	8
2.5. Generative Models for Algebraically Structured Data	10
3. PRELIMINARIES AND NOTATION	12
3.1. Mathematical Notation and Symbols	12
3.2. Core Definitions	13
4. METHODOLOGY	17
4.1. Classical GAN Approach	18
4.1.1. Dataset Construction	18
4.1.1.1. Numerical vs Symbolic Representation in Memory	19
4.1.1.2. Basis Selection: Monomial vs Bernstein Representation	20
4.1.1.3. Symbolic Dataset Diversification	26
4.1.2. GAN Training Setup	32
4.1.2.1. Latent Space and Input Representation	32
4.1.2.2. Model Architecture	33
4.1.2.3. Generator and Discriminator Architectures	34
4.1.2.4. Loss Functions and Optimization	35
4.1.2.5. Training Procedure and Parameters	36
4.1.2.6. Motivation for Architectural Simplicity	36
4.1.2.7. Variational Autoencoder Architecture and Training	36
4.2. GAN with Pretrained Discriminator	38
4.2.1. Dataset Construction	39
4.2.2. Binary Classification Training Setup	40
4.2.2.1. Objective	42
4.2.2.2. Method	42
4.2.2.3. Model Architecture	43
4.2.2.1. Objective	42
4.2.2.2. Method	42
4.2.2.3. Model Architecture	43

4.2.2.4.	Training and Evaluation	43
4.2.3.	Performance Metrics and Results of Binary Classification	44
4.2.3.1.	Results	46
4.2.4.	GAN Training Setup	47
4.2.4.1.	Results	50
5.	EXPERIMENTAL RESULTS	53
5.1.	Dataset Variants and Model Configurations	53
5.1.1.	Dataset Variants and Model Configurations for GAN	53
5.1.2.	Dataset Variants and Model Configurations for VAE	53
5.2.	Evaluation Metrics	54
5.2.1.	Evaluation Metrics for GAN	54
5.2.1.1.	Discriminator Accuracy	54
5.2.1.2.	Generator Accuracy	55
5.2.1.3.	Additional Evaluation Metrics	55
5.2.2.	Evaluation Metrics for VAE	56
5.3.	Experimental Results with GAN	56
5.3.1.	Interpreting Evaluation Metrics in GAN Training	56
5.3.2.	Preliminary Success: GANs for Degree-2 Real-Rooted Polynomials	58
5.3.3.	Motivation for Monitoring D and G Accuracy	61
5.3.4.	Main Experimental Results	61
5.3.4.1.	Visualizations and Comparative Graphs	62
5.4.	Experimental Results with VAE	66
5.4.1.	Latent Space Structure and Distribution of Root Types	66
5.4.2.	Real-Rooted Generation Success Rates	69
5.4.3.	Visualizations and Observations (VAE)	69
6.	CONCLUSION AND FUTURE WORK	73
6.1.	Conclusion	73
6.2.	Future Work	76
	REFERENCES	77
	BIOGRAPHY	80
	PUBLICATIONS WITHIN the SCOPE of THESIS STUDY	81
	APPENDICES	82
	Ek-A: Degree-2 GAN Experiment	82
	Ek-B: Detailed Visualization Analysis of GAN Experiments	83

LIST of ABBREVIATIONS and ACRONYMS

G	: Generator.
D	: Discriminator.
rrp	: Real rooted polynomial.
$non - rrp$	: Non real rooted polynomial.
d	: Degree of the polynomial.
x	: Independent variable of the polynomial.
$\mathbf{r}$	: Latent vector, input to the generator.
$\mathbf{b}$	: Output vector of the generator.
$\mathbf{a}$	: Coefficient vector of a rrp.
$\mathbf{u}$	: Input vector to the discriminator (either $\mathbf{a}$ or $\mathbf{b}$).
v	: Output of the discriminator.
J_D	: Discriminator loss function.
J_G	: Generator loss function.
η	: Learning rate in gradient-based optimization.
θ_G, θ_D	: Trainable parameters of the generator and discriminator networks.
$\mathcal{R}$	: Set of all real-rooted polynomials.
$\mathcal{P}_{\text{real}}$	: Dataset of real polynomials.
$\mathcal{P}_{\text{gen}}$	: Set of generated polynomials.
$\Delta(p)$	: Discriminant of polynomial p , used to assess the nature of its roots.
z	: A latent variable sampled from a prior distribution, typically $z \sim \mathcal{N}(0, I)$.
$q_\phi(z x)$	: The encoder function.
$p_\theta(x z)$	: The decoder function.
$\mathcal{L}_{\text{ELBO}}$	: The loss function for VAE.
$\text{KL}(\cdot \parallel \cdot)$	: Kullback–Leibler divergence.
$\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}$	: The training and test sets.
AI	: Artificial Intelligence
BCE	: Binary Cross Entropy
CSV	: Comma-Separated Values
D	: Discriminator
GAN	: Generative Adversarial Network
GPU	: Graphics Processing Unit
ID	: Identifier
JSON	: JavaScript Object Notation
ML	: Machine Learning
MLP	: Multi-Layer Perceptron
MSE	: Mean Squared Error
NN	: Neural Network
PDF	: Probability Density Function
ReLU	: Rectified Linear Unit
SGD	: Stochastic Gradient Descent
Tanh	: Hyperbolic Tangent Function
UI	: User Interface
VAE	: Variational Autoencoder

LIST OF FIGURES

	<u>Page</u>
Figure 4.1: Overview of the two alternative generative approaches.	17
Figure 4.2: Comparison of $W_{10}(x)$ and $\tilde{W}_{10}(x)$.	23
Figure 4.3: Root condition numbers of $W_{20}(x)$.	25
Figure 4.4: Scatter plot of symbolic irrational roots.	27
Figure 4.5: Kernel density plots of degree-2 polynomial roots.	28
Figure 4.6: Kernel density plots of degree-3 polynomial roots.	29
Figure 4.7: Kernel density plots of degree-4 polynomial roots.	30
Figure 4.8: Kernel density plots of degree-5 polynomial roots.	30
Figure 4.9: Root distribution for degree-3 polynomials with roots in $[-5, 5]$.	31
Figure 4.10: Root distribution for degree-4 polynomials with roots in $[-5, 5]$.	32
Figure 4.11: Schematic of our GAN architecture.	33
Figure 4.12: Generator (G) network architecture.	34
Figure 4.13: Discriminator (D) network architecture.	35
Figure 4.14: VAE structure.	37
Figure 4.15: Encoder and decoder layers.	37
Figure 4.16: Accuracy (left) and loss (right) for degree 3 polynomials.	44
Figure 4.17: Accuracy (left) and loss (right) for degree 4 polynomials.	45
Figure 4.18: Accuracy (left) and loss (right) for degree 5 polynomials.	45
Figure 4.19: GAN training results for degree 3.	51
Figure 4.20: GAN training results for degree 3(second run).	51
Figure 4.21: GAN training results for degree 4.	52
Figure 4.22: GAN training results for degree 5.	52
Figure 5.1: Number of non-rrp generated during test phase.	60
Figure 5.2: Accuracies of G and D for degree-2 polynomials.	63
Figure 5.3: Accuracies of G and D for degree-3 polynomials.	64
Figure 5.4: Generator accuracy curves across degrees 2 to 5 under both bases.	65
Figure 5.5: Latent space clustering for degree-2 polynomials using t-SNE.	67
Figure 5.6: Latent space clustering for degree-3 polynomials using t-SNE.	68
Figure 5.7: Latent space clustering for degree-4 polynomials using t-SNE.	68
Figure 5.8: Bar chart of average VAE success rates.	69
Figure 5.9: Box plot showing variance in VAE success rates.	70
Figure 6.1: Accuracy comparison between generator and decoder.	74
Figure 6.2: Number of non-real-rooted polynomials during test phase.	83
Figure 6.3: Accuracies of D and G over 20 epochs.	84
Figure 6.4: Accuracy trends of D and G over 20 epochs.	85
Figure 6.5: Accuracies of D and G over 10 epochs.	85
Figure 6.6: Seperate accuracy curves of D and G over 10 epochs.	86
Figure 6.7: Accuracies of D and G for degree-2.	87
Figure 6.8: Accuracies of D and G over 30 epochs.	87
Figure 6.9: Accuracies of D and G for degree-3.	88
Figure 6.10: Seperate accuracies ⁹¹ for degree-3.	89
Figure 6.11: Distribution of real roots in generated polynomials.	89
Figure 6.12: Discriminant distributions of generated polynomials.	90
Figure 6.13: Accuracies of G and D using monomial basis.	91

Figure 6.14: Accuracies of G and D using Bernstein basis.	91
Figure 6.15: Accuracies of D and G for degree-5 using monomial basis.	92
Figure 6.16: Accuracies of D and G for degree-5 using Bernstein basis.	93
Figure 6.17: Accuracy of G for degree-2 to degree-5.	93



LIST OF TABLES

	<u>Page</u>
Table 4.1: Comparison of numerical and symbolic representations.	20
Table 4.2: Dataset size and splits for each degree d .	41
Table 4.3: Discriminator performance for degree 3.	46
Table 4.4: Discriminator performance for degree 4.	46
Table 4.5: Discriminator performance for degree 5.	47
Table 4.6: Activation functions used in G and D.	49
Table 5.1: Quantitative summary of performance and success rates of G.	65
Table 5.2: Success rates of VAE-generated real-rooted polynomials.	71
Table 6.1: Numerical accuracy values corresponding to Figure 6.1.	75



1. INTRODUCTION

Mathematical object generation, particularly the synthesis of algebraic structures such as polynomials, has recently drawn significant interest from the machine learning community. The fundamental question motivating this work is whether deep generative models—specifically Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs)—can be effectively used to generate structured symbolic mathematical objects that satisfy predefined algebraic properties. This thesis focuses on the specific case of generating real-rooted univariate polynomials.

Real-rooted polynomials play a vital role in numerical analysis, control theory, and approximation theory, where the stability and interpretability of real solutions are essential. Despite their importance, existing generative models have largely focused on natural data types (images, text, audio) or unstructured symbolic data. The generation of symbolic mathematical expressions with explicit algebraic constraints—such as all roots being real—remains an open challenge.

The hypothesis of this thesis is as follows: With a carefully designed dataset and appropriate architectural choices, generative models like GANs and VAEs can learn to generate polynomials with only real roots. This hypothesis is built on two assumptions: (1) that real-rootedness can be implicitly captured through statistical patterns in symbolic coefficient space, and (2) that algebraically structured and diverse datasets improve the learnability of such constraints.

To investigate this hypothesis, we propose a two-pronged generative modeling framework involving both GANs and VAEs. First, we construct a set of symbolic polynomial datasets with controlled root types, basis representations (monomial and Bernstein), and symbolic rational/irrational encodings. Then, we train and evaluate both GAN and VAE architectures on these datasets. The VAE, in particular, is used to study the latent structure and disentanglement of polynomial root patterns. The GAN, on the other hand, is evaluated for its capacity to generate valid real-rooted instances through adversarial learning.

The main contributions of this thesis are as follows:

- A novel symbolic dataset construction framework capable of encoding rational and irrational roots exactly.
- A comparative study of monomial and Bernstein basis representations in terms of numerical stability and generative learnability.
- An empirical evaluation of GANs and VAEs for generating real-rooted polynomials across degrees and root types.
- Quantitative and qualitative analysis of generative success rates, including visualizations of latent spaces and output root structures.

This study bridges the gap between deep learning and symbolic mathematics, suggesting that algebraically constrained object generation is feasible within the generative modeling paradigm. It lays the groundwork for future exploration into algebra-aware learning systems and symbolic synthesis via deep generative models.

2. RELATED WORK

2.1. Classical Methods for Real-rooted Polynomial Generation

Real-rooted polynomials—those whose all roots lie on the real line—have been classically studied through symbolic algebraic methods. In this section, we review well-established techniques such as discriminant analysis, Sturm’s theorem, Descartes’ rule of signs, and the Routh–Hurwitz criterion. These approaches are contrasted with modern data-driven models such as GANs and VAEs to highlight the guarantees and limitations of each methodology.

Discriminant Criteria for Real Roots

The discriminant $\Delta(P)$ of a univariate polynomial $P(x)$ is an algebraic invariant that vanishes if and only if P has a repeated root. For quadratics and cubics, the sign and value of the discriminant provide a complete characterization of real roots:[1]. For higher-degree polynomials, however, a non-negative discriminant is a necessary but not sufficient condition for all roots to be real.

Sturm’s Theorem and Real Root Isolation

Sturm’s theorem uses a sequence of polynomial remainders to count the exact number of real roots of $P(x)$ in a given interval:[2]. Evaluating sign changes in the Sturm sequence at endpoints allows one to determine the number of distinct real roots, making it a deterministic and rigorous verification method.

Descartes’ Rule of Signs and Related Methods

Descartes’ rule offers an upper bound on the number of positive real roots based on the number of sign changes in the coefficients of $P(x)$. While not definitive, it is often combined with interval subdivision methods like Vincent’s Theorem for root isolation:[3].

Routh–Hurwitz Criterion and Stability Analysis

The Routh–Hurwitz criterion provides necessary and sufficient conditions for all roots of a polynomial to lie in the left half of the complex plane. Although it does not guarantee real-rootedness directly, it can imply the absence of complex roots in certain cases:[1].

Numerical Methods and Guarantees

Numerical solvers like Newton–Raphson or Jenkins–Traub can approximate all roots of a polynomial but lack formal guarantees of real-rootedness. Certified methods that combine symbolic and numeric checks are used to confirm real root counts:[4].

Classical vs. Data-driven Methods

Classical methods provide absolute guarantees based on mathematical theorems. In contrast, generative models like GANs and VAEs approximate real-rootedness by learning from data. Loss functions incorporating algebraic knowledge, such as discriminant-based penalties, improve the reliability of these models but do not guarantee correctness [5].

In this thesis, we specifically investigated whether classical algebraic constraints such as real-rootedness—typically ensured via discriminant analysis—could be mimicked within a generative learning framework. For degree-2 polynomials, we integrated the discriminant function into the loss of the generator and achieved a generation success rate above 99%. However, for higher-degree polynomials, the discriminant is not differentiable or computationally feasible, which required us to shift our focus to dataset design rather than constraint injection. Thus, our work explores not only the gap between classical and learning-based guarantees, but also how algebraic properties can be indirectly encoded through careful data modeling.

2.2. Machine Learning in Symbolic Mathematics

Recent advances in deep learning have prompted a growing interest in applying generative models to symbolic mathematical tasks. These studies explore whether machine

learning models can generate, manipulate, or learn properties of mathematical expressions such as polynomials, equations, or logical formulas. Below, we highlight several representative works and relate them to the goals of our thesis.

Generative Adversarial Networks for Symbolic Expressions.

Kreber and Hahn [6] introduced a Transformer-based GAN (TGAN-SR) for generating symbolic reasoning problems. Their approach shows that GANs can operate effectively on discrete symbolic data by leveraging transformer architectures and smoothed input representations. This supports our premise that GANs, if trained appropriately, can be used to generate polynomials with structural constraints such as having all real roots.

Variational Autoencoders for Expression Generation.

Popov et al. [7] proposed SEGVAE, a variational autoencoder for generating symbolic expressions tailored for symbolic regression. Their model uses guided decoding to ensure that generated expressions fit target datasets. Similarly, Mežnar et al. [8] developed a Hierarchical VAE that learns a latent space of expression trees, enabling efficient symbolic search. These works motivate our use of VAE architectures to learn latent representations of polynomials and generate real-rooted instances via decoding.

Symbolic Regression with Learned Priors.

Holt et al. [9] presented Deep Generative Symbolic Regression (DGSR), a method that uses pre-trained generative models as priors for symbolic equation discovery. Kamienny et al. [10] combined generative models with Monte Carlo Tree Search to improve symbolic regression accuracy. Both approaches highlight the benefit of combining generative capabilities with search mechanisms, which aligns with our attempt to interpret generator outputs through analytical or symbolic root verification.

Theoretical Perspectives.

Li [11] provides a theoretical framework for compositional generalization in neural networks. His results underscore the importance of preserving mathematical structure in

representations, which directly relates to the challenge of generating structured polynomials with real roots. Similarly, Ito et al. [12] argue that symbolic generalization should be benchmarked using algebraic complexity, which we interpret as an opportunity to assess generative models on increasingly complex polynomial families.

Relation to Our Work.

All these studies demonstrate the feasibility and limitations of applying machine learning to structured symbolic domains. Our work builds on these insights by focusing specifically on generating *real-rooted polynomials*, a family of objects with well-defined algebraic structure and mathematical importance. We implement and evaluate GAN and VAE models on synthetic polynomial datasets, assess their ability to preserve root reality, and analyze their limitations across polynomial degrees and basis representations.

2.3. Generative Models for Mathematical Object Synthesis

Recent years have witnessed increasing interest in using generative models such as GANs and VAEs to synthesize mathematical expressions, including polynomials and algebraic equations. Unlike images or natural language, these symbolic expressions are discrete and structurally constrained, requiring model designs that ensure syntactic and semantic correctness. Below, we review recent literature that inspired our approach to generating real-rooted polynomials.

Transformer-GANs for Symbolic Formulas

Kreber and Hahn [13] proposed a Transformer-based GAN to generate well-formed symbolic expressions such as algebraic or logical formulas. By introducing Gaussian noise into one-hot encodings, they relaxed the discrete nature of the input, allowing stable GAN training. Moreover, they introduced an objective term based on classifier uncertainty to bias generation toward complex, solver-challenging expressions. Their work demonstrates that symbolic mathematical structures can be learned and generated adversarially.

Relevance to Our Work: Their method inspires our use of custom GAN objectives

(e.g., real-root encouragement), showing how symbolic properties can be promoted during adversarial training.

HVAE: Hierarchical VAE for Tree-Based Expressions

Mežnar et al. [14] introduced HVAE, a hierarchical VAE designed for tree-structured symbolic expressions. By representing equations as parse trees, the model enforces syntactic validity and yields a smooth latent space from which valid expressions can be decoded or evolved. This structure-aware encoding and decoding process allows for expressive yet semantically grounded generation.

Relevance to Our Work: HVAE’s latent space design is conceptually aligned with our effort to generate valid polynomials. A similar architecture could enforce polynomial degree and root symmetry constraints.

Graph-EQ: VAE with Graph Neural Networks

Ranasinghe et al. [15] proposed Graph-EQ, a graph-based conditional VAE for generating symbolic expressions. Mathematical formulas are modeled as directed acyclic graphs (DAGs), and the model is trained to both reconstruct and optimize expressions for data fit. The latent space exploration led to rediscovery of benchmark equations.

Relevance to Our Work: Graph-based representations of polynomials (as expression DAGs) could enhance structure preservation in generation. Latent optimization strategies can help enforce the real-root property.

SEGVAE: Generative Modeling with Domain Constraints

Popov et al. [16] designed SEGVAE, a VAE model that incorporates fast-check predicates to guide the decoder toward expressions satisfying domain constraints. Their approach mixes reconstruction loss with performance-based rewards to enforce semantic properties.

Reinforcement Learning for Polynomial Synthesis

Ståhl et al. [17] investigate the generation of polynomial expressions via reinforcement learning. The proposed framework formulates polynomial construction as a sequential decision-making task, where an agent incrementally selects terms to optimize an external evaluation metric. The goal is to generate polynomial models that explain observed numerical data.

Although the approach succeeds in producing interpretable models for small-scale data, it struggles with controlling algebraic properties such as root multiplicity or real-rootedness. In particular, they note a low success rate in generating well-structured or algebraically valid polynomials, especially for higher degrees. Their results suggest that even non-adversarial generative approaches like RL encounter challenges when symbolic algebraic constraints are involved.

These limitations resonate with the challenges encountered in our GAN-based architecture. Despite architectural differences, both frameworks highlight the fundamental difficulty of generating structurally valid symbolic objects with predefined algebraic properties.

Relevance to Our Work: We take methodological inspiration from this design by incorporating root-realness constraints or discriminant-based checks into the VAE training process.

Summary

Each study contributes architectural or training innovations suitable for symbolic domains. By integrating structural encodings, custom losses, and latent space strategies, these works provide conceptual and methodological foundations for our approach. Our thesis builds upon these by tailoring the generation process specifically for real-rooted polynomials using Bernstein and monomial bases.

2.4. Polynomial Basis and Learnability

The choice of basis for representing polynomials has significant implications for both numerical stability and the learnability of models that aim to generate or approximate

polynomial functions. This section explores how the monomial and Bernstein bases compare in terms of stability, root sensitivity, and their suitability in generative modeling.

Monomial vs. Bernstein Basis

A polynomial $p(x)$ of degree n can be represented in the monomial (power) basis as:

$$p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n, \quad (2.1)$$

whereas in the Bernstein basis, it takes the form:

$$p(x) = \sum_{k=0}^n b_k B_{k,n}(x), \quad \text{where } B_{k,n}(x) = \binom{n}{k} x^k (1-x)^{n-k}. \quad (2.2)$$

The monomial basis suffers from poor numerical conditioning, especially for high degrees. In contrast, the Bernstein basis is optimally stable [18, 19].

Implications for Learnability

Learning generative models over polynomial coefficients can be adversely affected by the instability of the monomial basis. In our experiments, we observed that VAEs and GANs trained on monomial coefficients exhibited unstable loss behavior. Bernstein coefficients, on the other hand, led to more consistent and stable training, with smoother gradient flows and less overfitting.

Related Work

Normalizing Flows. Bernstein basis polynomials have been successfully used in normalizing flows [20, 21]. These works demonstrate improved stability and expressiveness.

Neural Network Robustness. Bernstein activations have been proposed in DeepBern-Nets for improved certified robustness [22].

Graph Neural Networks. The Bernstein basis has been used to parametrize spectral filters in GNNs [23].

Shape Modeling. In airfoil generation tasks, using Bézier curves (Bernstein-based) has shown improved control and interpretability compared to monomials [24].

Relevance to Real-rooted Polynomial Generation

The generation of real-rooted polynomials benefits from the use of bases like Bernstein. Stability ensures that the roots do not oscillate under small coefficient perturbations. Moreover, known properties of the Bernstein basis can be exploited to enforce real-rootedness via constraints [19].

Conclusion. Theoretical and empirical evidence strongly suggests that using the Bernstein basis in generative models provides improved numerical stability, better learnability, and more robust real-rooted polynomial generation.

2.5. Generative Models for Algebraically Structured Data

In recent years, generative models have been adapted for tasks involving algebraic and symbolic structures, including polynomial expressions. These models, particularly Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs), have demonstrated potential in capturing underlying mathematical regularities and generating syntactically valid expressions. Below, we review significant works and analyze their relevance to our study on generating real-rooted polynomials.

VAEs for Symbolic Expression Generation

Symbolic Expression VAE (Popov et al., 2023): This model employs domain-specific constraints and fast-check predicates to guide the generation of valid mathematical expressions [16]. It highlights the effectiveness of integrating symbolic validity checks into the training loop.

Hierarchical VAE (Mežnar et al., 2023): This approach encodes and decodes hierarchical parse trees of expressions, resulting in a smooth latent space suitable for exploration and optimization [8].

Grammar and Syntax-Constrained VAEs: Earlier works like GVAE and SD-VAE enforce syntactic validity using grammar constraints, which can be too rigid but informative for symbolic generation [25].

GANs for Symbolic Data

Transformer-based Symbolic GANs (Kreber & Hahn, 2023): By leveraging Transformers and noise injection, this architecture generates valid symbolic problems, showing the feasibility of GANs in discrete symbolic domains [26].

Reinforcement and Hybrid Methods

Reinforcement Learning (Ståhl et al., 2021): Here, the generator is trained via reward signals to produce interpretable polynomials that approximate data well, aligning with our goal of structure-aware generation [17].

Hybrid Search-Generative Systems: Neural generators coupled with symbolic search (e.g., Monte Carlo Tree Search) create a synergistic framework for producing algebraically valid structures.

These studies show that symbolic validity, structural encoding, and latent space exploration are key to successful generation. Our work integrates these insights into a GAN/VAE system targeting real-rooted polynomial generation.

3. PRELIMINARIES AND NOTATION

This chapter introduces the essential mathematical concepts, notational conventions, and terminology used throughout the thesis. It aims to provide a unified and consistent framework for representing polynomials, symbolic data structures, and generative models.

3.1. Mathematical Notation and Symbols

The following symbols and conventions will be used throughout this thesis:

- d — Degree of the polynomial.
- x — Independent variable of the polynomial.
- $\mathbf{r} \in \mathbb{R}^{d+1}$ — Latent vector sampled uniformly from $[-1, 1]^{d+1}$ and input to the generator.
- $\mathbf{b} = G(\mathbf{r}) \in \mathbb{R}^{d+1}$ — Output vector from the generator representing polynomial coefficients (fake data).
- $\mathbf{a} = [a_0, a_1, \dots, a_d] \in \mathbb{R}^{d+1}$ — Coefficient vector of a real polynomial sampled from the symbolic dataset (real data).
- $\mathbf{u} \in \mathbb{R}^{d+1}$ — Input vector to the discriminator (either $\mathbf{a}$ or $\mathbf{b}$).
- $v = D(\mathbf{u}) \in [0, 1]$ — Output of the discriminator, representing the probability of the input being real.
- J_D — Discriminator loss function.
- J_G — Generator loss function.
- η — Learning rate in gradient-based optimization.
- θ_G, θ_D — Trainable parameters of the generator and discriminator networks.
- $\mathcal{R}$ — Set of all real-rooted polynomials.
- $\mathcal{P}_{\text{real}}$ — Dataset of real polynomials.
- $\mathcal{P}_{\text{gen}}$ — Set of generated polynomials.
- $\Delta(p)$ — Discriminant of polynomial p , used to assess the nature of its roots.

- z : A latent variable sampled from a prior distribution, typically $z \sim \mathcal{N}(0, I)$.
- $q_\phi(z|x)$: The encoder function, parameterized by ϕ , approximates the posterior distribution of the latent variable given the input.
- $p_\theta(x|z)$: The decoder function, parameterized by θ , models the likelihood of reconstructing x from z .
- $\mathcal{L}_{\text{ELBO}}$: The loss function for VAE, which is the Evidence Lower Bound:

$$\mathcal{L}_{\text{ELBO}}(x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \text{KL}(q_\phi(z|x) \parallel \mathcal{N}(0, I))$$

- $\text{KL}(\cdot \parallel \cdot)$: Kullback–Leibler divergence, a measure of difference between two probability distributions.
- $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}$: The training and test sets used to learn and evaluate generative models, respectively.

3.2. Core Definitions

Polynomial

A *polynomial* in one variable is a function of the form:

$$p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n = \sum_{i=0}^n a_i x^i, \quad (3.1)$$

where $a_0, a_1, \dots, a_n \in \mathbb{R}$ and $a_n \neq 0$. The integer n is called the *degree* of the polynomial [27].

Real-Rooted Polynomial

A polynomial $p(x)$ with real coefficients is said to be *real-rooted* if all its roots lie in the real number set $\mathbb{R}$. That is:

$$p(x) = 0 \Rightarrow x \in \mathbb{R}. \quad (3.2)$$

This is an important property in algebra, combinatorics, and control theory [28].

Discriminant

The *discriminant* of a degree- n polynomial $f(x) = a_0x^n + a_1x^{n-1} + \dots + a_n$ with roots $\alpha_1, \dots, \alpha_n$ is defined as:

$$\Delta(f) = a_0^{2n-2} \prod_{1 \leq i < j \leq n} (\alpha_i - \alpha_j)^2. \quad (3.3)$$

It vanishes if and only if the polynomial has a repeated root [29].

Monomial Basis

The *monomial basis* (or standard basis) for the space of polynomials of degree at most n is given by:

$$\mathcal{B}_{\text{mono}} = \{1, x, x^2, \dots, x^n\}. \quad (3.4)$$

Any polynomial $p(x)$ of degree $\leq n$ can be uniquely written as:

$$p(x) = \sum_{i=0}^n a_i x^i, \quad (3.5)$$

with coefficient vector $\mathbf{a} = [a_0, a_1, \dots, a_n]$ [30].

Bernstein Basis

The *Bernstein basis* of degree n over $[0, 1]$ is defined as:

$$B_{v,n}(x) = \binom{n}{v} x^v (1-x)^{n-v}, \quad v = 0, 1, \dots, n. \quad (3.6)$$

These basis functions are non-negative, symmetric, and form a partition of unity:

$$\sum_{v=0}^n B_{v,n}(x) = 1, \quad \forall x \in [0, 1]. \quad (3.7)$$

They are widely used due to their numerical stability [30].

Latent Space

In generative models, the *latent space* is a continuous vector space $\mathbb{R}^d$ from which input vectors $\mathbf{r} \sim p(\mathbf{r})$ are sampled. The generator maps these to output instances [31].

Generator

The *generator* G is a neural network that maps a latent input $\mathbf{r} \in \mathbb{R}^{d+1}$ to a polynomial coefficient vector $\mathbf{b} = G(\mathbf{r}) \in \mathbb{R}^{d+1}$ [31].

Discriminator

The *discriminator* D is a neural network that receives an input vector $\mathbf{u} \in \mathbb{R}^{d+1}$ and outputs a scalar $v = D(\mathbf{u}) \in [0, 1]$, indicating the probability of $\mathbf{u}$ being real [31].

Value Function

The *value function* in GANs defines a two-player minimax game:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{\mathbf{r} \sim p(\mathbf{r})} [\log(1 - D(G(\mathbf{r})))] \quad [32]. \quad (3.8)$$

G tries to minimize this objective, while D tries to maximize it.

Multi-Layer Perceptron (MLP)

An *MLP* is a fully-connected feedforward neural network with one or more hidden layers, where each neuron applies a non-linear activation function [33].

Loss Function

A *loss function* $L(\theta)$ measures the difference between predicted and true values. It is used to update parameters via gradient descent [33].

Stochastic Gradient Descent (SGD)

SGD updates parameters using gradients computed from random mini-batches:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} L(\theta; \text{batch}) \quad [34]. \quad (3.9)$$

Epoch and Batch

An *epoch* is one complete pass through the training dataset. A *batch* is a subset of data used to compute a single update [33].

The notation and definitions above will serve as the foundational language for all subsequent chapters.

Generative Adversarial Network (GAN)

A Generative Adversarial Network (GAN) [35] consists of two neural networks trained in competition:

- **Generator (G):** Learns a mapping from a random latent vector $z \sim \mathcal{N}(0, I)$ to a structured output. In our context, G aims to generate coefficient vectors $a = [a_0, a_1, \dots, a_d]$ corresponding to real-rooted polynomials.
- **Discriminator (D):** A binary classifier that distinguishes between real coefficient vectors (sampled from the dataset) and fake ones produced by G . It outputs a probability $D(a) \in [0, 1]$ indicating the likelihood that the input is real.

The two networks are trained via a minimax optimization problem [35]:

$$\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim \mathcal{N}(0, I)} [\log(1 - D(G(z)))] \quad (3.10)$$

Variational Autoencoder (VAE)

A Variational Autoencoder (VAE) [36] is a generative model composed of two key modules:

- **Encoder ($q_\phi(z|x)$):** Maps the input coefficient vector $x = a = [a_0, \dots, a_d]$ to a latent representation z via a learned probabilistic distribution, typically modeled as a multivariate Gaussian with mean and variance output by the encoder.
- **Decoder ($p_\theta(x|z)$):** Attempts to reconstruct the input from a sample z drawn from the encoded latent distribution. In our setting, the decoder outputs a coefficient vector a such that the reconstructed polynomial is real-rooted.

The VAE is trained by maximizing the Evidence Lower Bound (ELBO) [36]:

$$\mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z)] - \text{KL}(q_\phi(z|x) \parallel \mathcal{N}(0, I)) \quad (3.11)$$

4. METHODOLOGY

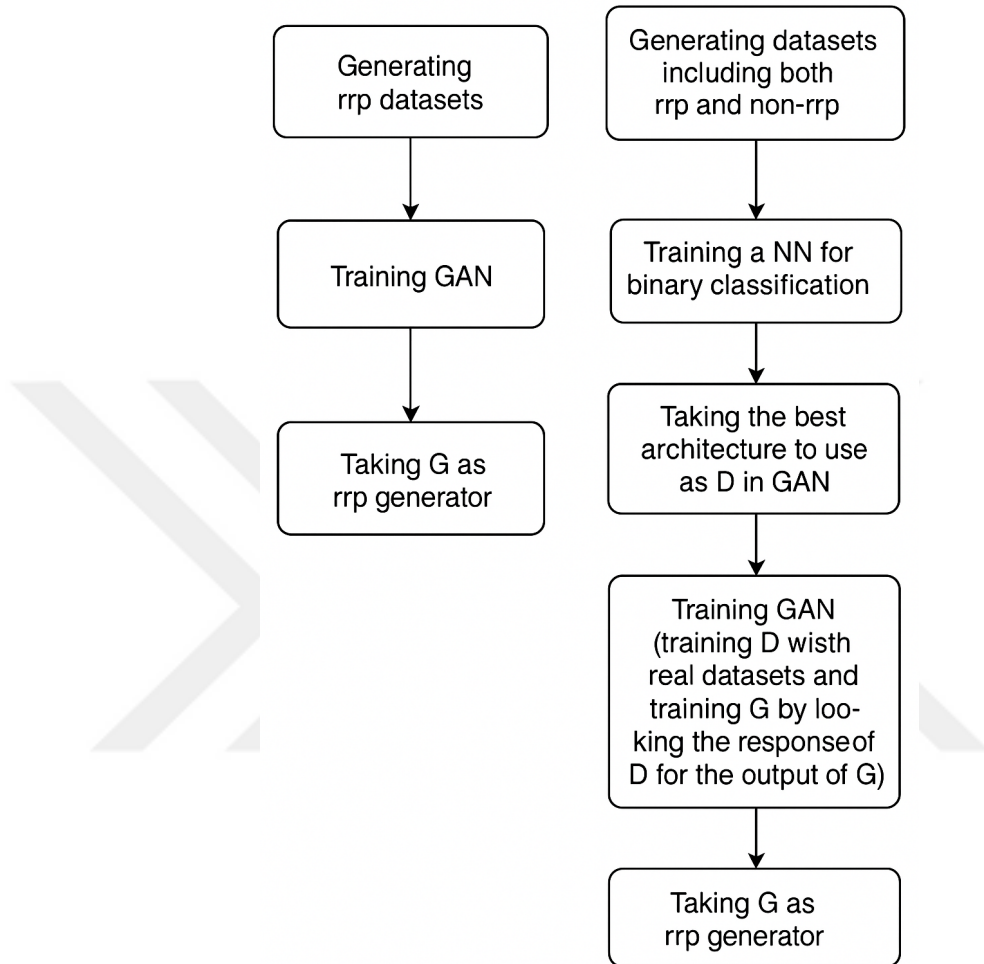


Figure 4.1. Overview of the two alternative generative approaches investigated in this thesis.

In this chapter, we present two complementary generative approaches for producing real-rooted polynomials:

- The first approach injects classical algebraic constraints directly into the generator loss.
- The second approach utilizes a separately trained binary classifier as a discriminator in a GAN setup.

4.1. Classical GAN Approach

This section presents the overall architecture and training procedure of the first proposed generative model. It includes the structure of the generator and discriminator networks, the loss functions, and the details of how real-rootedness is encouraged during training.

4.1.1. Dataset Construction

A major contribution of this thesis is the construction of custom symbolic datasets specifically designed for training GAN models to generate real-rooted polynomials.

The full codebase used to generate these datasets is available at this link below:

<https://gitlab.com/BasakKarakas/thesis-process>

Initially, numerical datasets were created by sampling d real-valued roots $\{x_1, x_2, \dots, x_d\}$ uniformly from the interval $(-1, 1)$, using the `rand()` function in the Julia programming language. A polynomial of degree d was constructed by multiplying the linear factors:

$$p(x) = \prod_{i=1}^d (x - x_i) \quad (4.1)$$

For example, for degree 2, let $x_1 = 0.25$, $x_2 = -0.7$. Then:

$$p(x) = (x - 0.25)(x + 0.7) = x^2 + 0.45x - 0.175 \quad (4.2)$$

$$\Rightarrow \mathbf{a} = [-0.175, 0.45, 1] \quad (4.3)$$

For degree 3, let $x_1 = -0.5$, $x_2 = 0.1$, $x_3 = 0.8$. Then:

$$p(x) = (x + 0.5)(x - 0.1)(x - 0.8) = x^3 - 0.4x^2 - 0.09x + 0.04 \quad (4.4)$$

$$\Rightarrow \mathbf{a} = [0.04, -0.09, -0.4, 1] \quad (4.5)$$

This process was repeated for degrees ranging from $d = 2$ to $d = 10$. For each generated polynomial, the coefficient vector $\mathbf{a} = [a_0, a_1, \dots, a_d]$ was recorded. The same GAN architecture was trained separately for each degree using these numerically generated datasets.

Initial training experiments using numerically generated datasets showed that the GAN architecture failed to learn accurately. One suspected cause was the use of floating-point approximations to represent roots and coefficients in memory. For instance, a rational number such as $\frac{10}{3}$ would be stored approximately as 3.3333333333333335 in 64-bit floating-point format, and irrational numbers like $\sqrt{2}$ cannot be represented exactly at all. These approximations may lead to inconsistencies between the intended and actual root structures of the polynomials.

To resolve this issue, all roots and coefficients were instead represented symbolically using the SymPy library. In symbolic form, numbers are stored as algebraic objects such as `Rational(10, 3)` or `sqrt(2)`, preserving their exact form during computation. This symbolic structure allowed for precise representation of both rational and irrational roots. All dataset generation, storage, and training operations were conducted entirely using symbolic representations. Each dataset was saved as a `.jls` file, containing 1,000 training and 200 testing polynomials per configuration.

4.1.1.1. Numerical vs Symbolic Representation in Memory

In machine learning models, the internal representation of numerical data can significantly affect training quality. In this study, we observed that numerical inaccuracies introduced by floating-point representation may be one of the reasons behind the failure of early GAN models to learn real-rooted polynomial structures accurately.

Floating-Point (Numerical) Representation: Real numbers are typically stored in computers using the IEEE 754 floating-point standard. In double precision (float64), only a limited number of decimal values can be exactly represented. For example:

```
>>> 10 / 3
3.3333333333333335
>>> 0.1 + 0.2
0.30000000000000004
```

Even the seemingly simple value 0.1 cannot be stored precisely, because its binary representation is infinitely repeating. These approximations can lead to rounding errors and cumulative distortions, especially when modeling mathematical structures such as

polynomials.

Symbolic Representation (SymPy): Using the SymPy library, numbers are stored as algebraic objects rather than decimal approximations. For example:

```
>>> Rational(10, 3)
10/3
>>> sqrt(2)
sqrt(2)
```

These symbolic objects preserve the exact structure of the numbers and support algebraic operations without introducing rounding errors. For this reason, symbolic representation was adopted throughout the dataset construction and GAN training pipeline to ensure mathematical integrity.

Table 4.1. Comparison of numerical and symbolic representations.

Feature	Numerical (float64)	Symbolic (SymPy)
Precision	Approximate	Exact
Rational support	Approximate (e.g., 3.333...)	Exact (e.g., 10/3)
Irrational support	Approximate (e.g., 1.414...)	Exact (e.g., $\sqrt{2}$)
Representation	IEEE 754 bit string	Object (Class-based)
Arithmetic operations	Rounded results	Exact algebraic results

4.1.1.2. Basis Selection: Monomial vs Bernstein Representation

Monomial Basis. The monomial (or power) basis is the standard polynomial basis defined by the set of monomials:

$$\mathcal{B}_{\text{mono}} = \{1, x, x^2, \dots, x^d\} \quad (4.6)$$

A univariate polynomial of degree d over the real numbers can be expressed in this

basis as:

$$p(x) = \sum_{i=0}^d a_i x^i = a_0 + a_1 x + a_2 x^2 + \cdots + a_d x^d \quad (4.7)$$

Here, the coefficient vector $\mathbf{a} = [a_0, a_1, \dots, a_d] \in \mathbb{R}^{d+1}$ uniquely determines the polynomial in the monomial basis. This basis is simple and algebraically convenient but, as discussed below, may be numerically unstable for certain tasks such as interpolation or root-finding, especially for large d .

Numerical Instability of the Monomial Basis

Despite its algebraic simplicity, the monomial (or power) basis is known to be numerically unstable for high-degree polynomials [37, 38]. In this basis, the polynomial is expressed as:

$$p(x) = \sum_{i=0}^d a_i x^i \quad (4.8)$$

As the degree d increases, the monomial basis functions x^i tend to become nearly linearly dependent over finite intervals, particularly on $[a, b]$. This behavior causes the associated Vandermonde matrices to become increasingly ill-conditioned, which amplifies the effects of round-off errors or small perturbations in the coefficients.

This instability is especially problematic in root-finding tasks. Even minute changes in the coefficients—arising from data imprecision or floating-point representation—can lead to significant shifts in the polynomial's roots, a phenomenon known as *root drift* [39]. Consequently, the monomial basis is often avoided in favor of more numerically stable alternatives in computational mathematics.

Wilkinson Polynomial: A Classic Case of Instability

A striking illustration of the numerical instability inherent in the monomial basis is provided by the Wilkinson polynomial [39]. The degree- n Wilkinson polynomial is defined as:

$$W_n(x) = \prod_{k=1}^n (x - k) \quad (4.9)$$

It has n simple real roots at the integers $1, 2, \dots, n$, and is often used as a benchmark example in numerical analysis. When expressed in the monomial basis, however, the coefficients of $W_n(x)$ grow rapidly in magnitude, leading to severe ill-conditioning.

Wilkinson famously demonstrated that a minuscule perturbation in just one coefficient

of the monomial expansion of $W_{20}(x)$ —such as modifying the coefficient of x^{19} by as little as 10^{-7} —can cause drastic changes in the roots. In particular, some of the real roots may become complex, and others may drift far from their original locations:

$$x = 15 \quad \longrightarrow \quad x \approx 15.03 \pm 0.12i \quad (4.10)$$

This phenomenon, often referred to as *root drift*, highlights how numerically fragile high-degree polynomials can be when represented in the power basis. Even very small coefficient errors, arising from floating-point rounding or data noise, can lead to qualitatively incorrect root structures.

Wilkinson’s example remains a classic demonstration of the dangers of using the monomial basis for numerical computations involving polynomial roots.

As a concrete example, we consider the Wilkinson polynomial of degree 10:

$$W_{10}(x) = \prod_{k=1}^{10} (x - k) \quad (4.11)$$

To illustrate the instability of the monomial basis, we construct a perturbed polynomial by modifying the coefficient of the x^9 term. Specifically, we define:

$$\tilde{W}_{10}(x) = W_{10}(x) + 10^{-4}x^9 \quad (4.12)$$

This minimal change is applied to observe its effect on the location and nature of the roots.

Figure 4.2 demonstrates the effect of a small perturbation applied to the coefficient of x^9 in the Wilkinson polynomial of degree 10. While the original polynomial has real roots at every integer from 1 to 10 (shown as blue dots), the perturbed polynomial exhibits notable deviations (red dots). Most significantly, the roots near 7 through 10 are no longer real and thus disappear from the real axis — indicating that they have turned into complex conjugate pairs. This transformation is evident by the absence of red markers corresponding to those positions. Despite the perturbation being as small as 10^{-4} , such drastic changes in root locations — including real-to-complex conversion — reveal the severe numerical instability of the monomial basis. Even slight coefficient inaccuracies in high-degree polynomials can result in the complete loss or displacement of roots.

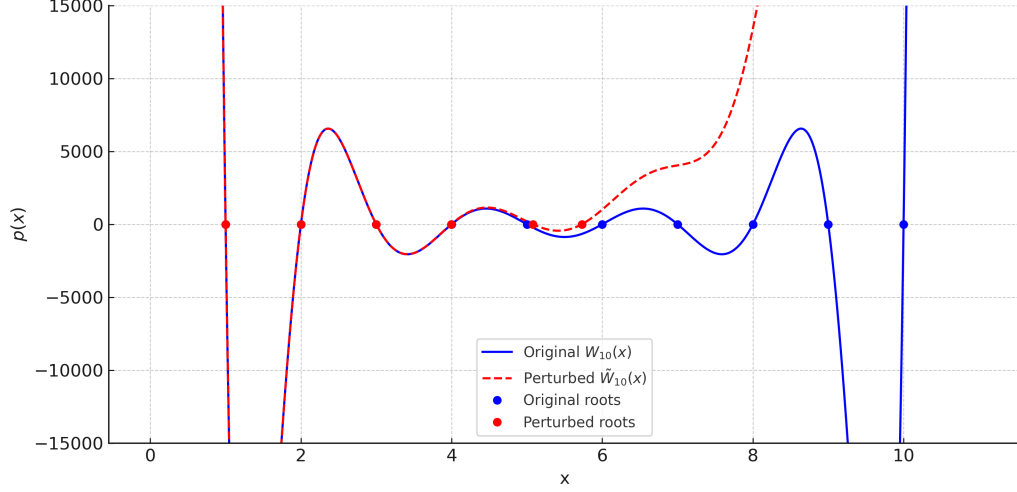


Figure 4.2. Comparison of $W_{10}(x)$ and $\tilde{W}_{10}(x)$ under small coefficient perturbation.

The Wilkinson example above highlights the extreme sensitivity of polynomials expressed in the monomial (power) basis. Even a negligible change in a single coefficient resulted in a drastic alteration of the root structure, including the formation of complex roots. This phenomenon, known as root drift, is well-documented and has motivated the search for alternative representations that offer greater numerical robustness. One such alternative is the Bernstein polynomial basis, which has been shown to exhibit significantly better stability properties.

Bernstein Basis. The Bernstein basis polynomials of degree n are defined over the interval $[0, 1]$ as follows:

$$B_k^n(x) = \binom{n}{k} x^k (1-x)^{n-k}, \quad \text{for } k = 0, 1, \dots, n \quad (4.13)$$

The set $\{B_0^n(x), B_1^n(x), \dots, B_n^n(x)\}$ forms a basis for the space of polynomials of degree at most n . Any polynomial $p(x)$ of degree n can be written in this basis as:

$$p(x) = \sum_{k=0}^n b_k B_k^n(x) \quad (4.14)$$

where $b_k \in \mathbb{R}$ are the Bernstein coefficients. These basis polynomials possess several favorable properties on the interval $[0, 1]$, including positivity, symmetry, and partition of unity:

- $B_k^n(x) > 0$ for all $x \in (0, 1)$,

- $\sum_{k=0}^n B_k^n(x) = 1$ for all $x \in [0, 1]$,
- $B_k^n(x) = B_{n-k}^n(1 - x)$ for all $x \in [0, 1]$.

These properties make the Bernstein basis particularly suitable for stable numerical computations, such as polynomial evaluation and root analysis [40].

Remark on Domain Consistency. Although the classical Bernstein basis is defined over the interval $[0, 1]$, the polynomial datasets used in this study involve root distributions over intervals such as $[-1, 1]$, $[-2, 2]$, and $[-5, 5]$. In the experiments, the Bernstein basis was applied directly to these domains without performing an affine transformation to map them into the standard interval.

This methodological decision was made in order to preserve the structure of the original polynomials and allow for consistent comparisons across basis representations. However, it should be noted that several favorable properties of the Bernstein basis—such as positivity, partition of unity, and stability—are formally guaranteed only on the interval $[0, 1]$. Therefore, applying the Bernstein basis outside this range may reduce its theoretical advantages in terms of numerical behavior.

Future improvements may involve normalizing all polynomial domains to $[0, 1]$ prior to basis conversion to fully leverage the well-established numerical properties of the Bernstein basis.

Stability Comparison via Root Condition Numbers. The sensitivity of a polynomial's roots with respect to small perturbations in its coefficients can be quantified by the **root condition number**. Given a simple root x_0 of a polynomial $p(x)$, its condition number is defined as follows:

$$\kappa(x_0) = \left| \frac{x_0 \cdot p'(x_0)}{p(x_0)} \right| \quad (4.15)$$

Intuitively, this quantity measures how much a small relative error in the coefficients of the polynomial can affect the computed value of the root. A high condition number implies that the root is more sensitive and potentially unstable, while a low condition number indicates numerical robustness.

To illustrate this concept, we refer to the study of Farouki and Rajan [?], where they compare the condition numbers of the roots of the degree-20 Wilkinson polynomial:

$$W_{20}(x) = \prod_{i=1}^{20} (x - i) \quad (4.16)$$

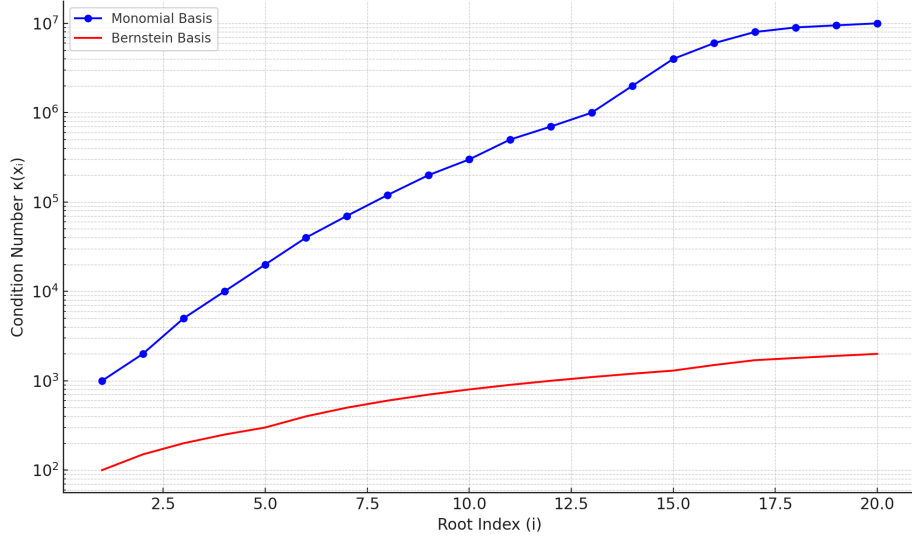


Figure 4.3. Root condition numbers of $W_{20}(x)$ in monomial and Bernstein bases.

Figure 4.3 shows the logarithmic condition numbers of the 20 real roots of $W_{20}(x)$, computed in both the monomial and Bernstein representations. As observed, the monomial basis leads to extremely high condition numbers (exceeding 10^7) for several roots, especially near the endpoints. In contrast, the Bernstein basis results in significantly smaller and more uniform values (typically $10^2 - 10^3$). This example demonstrates how Bernstein representation provides much greater numerical stability under coefficient perturbations in high-degree polynomials.

Symbolic Irrational Root Construction As part of the symbolic dataset construction, a specific method was developed to generate irrational roots with full algebraic traceability and bounded values. The following section outlines this process.

To generate irrational roots symbolically within a desired interval, a custom Julia function named `randIrrational(lowerBound, upperBound, numOfTerm, rootDegree)` was implemented. This function produces a symbolic expression composed of up to `numOfTerm` algebraic terms, each of the form $\pm \frac{b\sqrt[n]{n}}{d}$, where the root degree b is sampled

randomly from the range $[1, \text{rootDegree}]$, and n, d are positive integers. The sign of each term is also sampled randomly from $\{-1, 1\}$, and the overall sum is checked to ensure it lies within the interval $[\text{lowerBound}, \text{upperBound}]$. If the sum falls outside this range, it is rejected and the sampling process is repeated.

This design enables the construction of irrational numbers that are algebraic, human-readable, and symbolically traceable. The use of exact root expressions allows full preservation of symbolic structure during polynomial generation, eliminating any numerical approximation errors. The resulting values are suitable for precise root-to-coefficient conversion and downstream symbolic operations.

As an example, the call `randIrrational(-2.0, 2.0, 5, 5)` generates a symbolic irrational number within the interval $[-2, 2]$, constructed as the sum of at most five algebraic terms, each with a root degree no greater than 5. One possible output of this function is:

$$-\frac{\sqrt[4]{94917310840249}}{5000} + \frac{\sqrt{8072645247813783}}{10000000} + \frac{2629879938366979}{2500000000000000} \quad (4.17)$$

This expression combines nested radicals and rational fractions in symbolic form. It satisfies the constraints imposed on degree and value bounds, and it can be used directly in symbolic root-based polynomial construction. Notably, all operations preserve exactness and remain fully compatible with symbolic algebra systems such as SymPy.

4.1.1.3. Symbolic Dataset Diversification

Motivated by the numerical stability advantages of the Bernstein basis and the structural fragility observed in monomial representations, this study adopted a symbolic framework for polynomial dataset construction. All roots and coefficients were encoded symbolically, allowing exact arithmetic, precise root placement, and robust algebraic manipulation during model training.

To evaluate how root structure and basis representation affect generative model performance, the symbolic datasets were diversified along four main dimensions:

- Polynomial degree: 2 through 5
- Root range: $[-1, 1]$, $[-2, 2]$, and $[-5, 5]$
- Root type: rational only, irrational only, mixed real roots

- Basis: Monomial and Bernstein

This systematic variation yielded 56 distinct datasets, each comprising 1,000 symbolic polynomials for training and 200 for testing.

The coefficient vectors were stored as symbolic arrays $\mathbf{a} = [a_0, a_1, \dots, a_n]$, without rounding or numeric approximation, and directly used during the GAN training process. The resulting datasets enabled controlled experiments on how representational and structural features of polynomials influence learning dynamics under symbolic precision.

Root Distribution Visualization. This section presents visual analyses of the symbolic root generation process, focusing on how the roots are distributed across different sampling configurations.

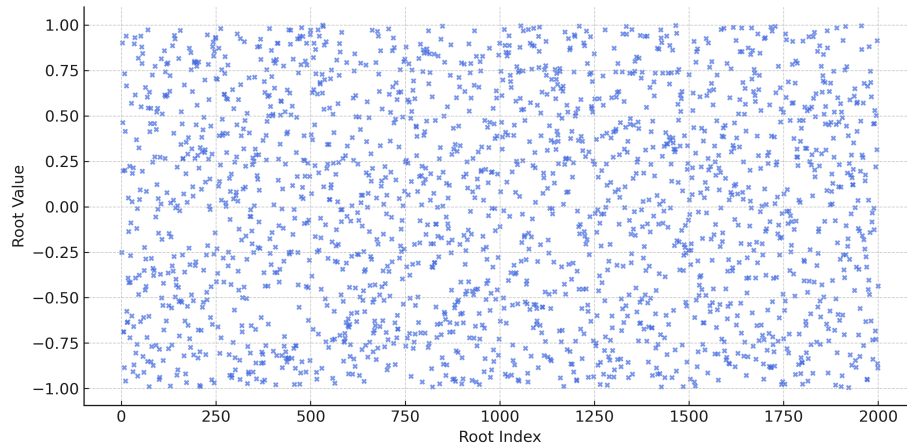


Figure 4.4. Scatter plot of symbolic irrational roots.

To visualize the effectiveness of the irrational root generation process, we extracted the symbolic roots of 1000 polynomials of degree 2. Since each polynomial contains two roots, the total of 2000 values was flattened and plotted in Figure 4.4. Each dot corresponds to a single irrational root. The values appear well-distributed within the expected interval $[-1, 1]$, without visible clustering.

To explore how different sampling types affect the overall root distribution, we generated density plots based on symbolic polynomials with rational-only, irrational-only,

and arbitrary real roots. As shown in Figure 4.5, each root type leads to a distinct shape in the kernel density estimation. While rational roots tend to be more uniformly distributed, irrational roots form a multimodal pattern. The mixed real case appears structurally in-between, influenced by both rational and irrational terms.

These visualizations validate that the symbolic generation framework produces structurally varied yet bounded root sets based on the user-defined constraints.

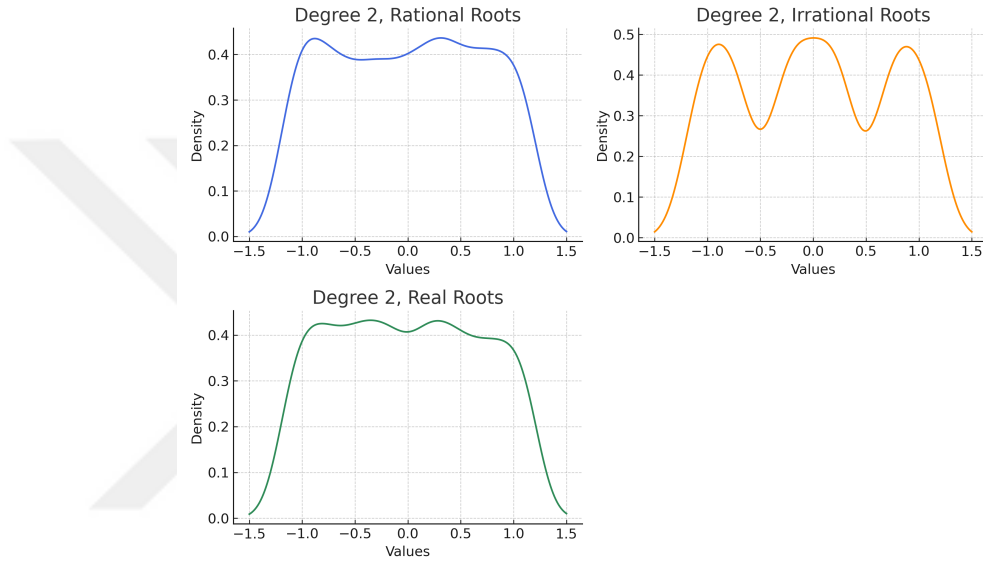


Figure 4.5. Kernel density plots of degree-2 polynomial roots.

Figure 4.5 shows that rational roots are more uniformly distributed, while irrational ones form a multimodal structure, with real roots falling in between—highlighting the structural distinctions induced by symbolic root generation.

The distributional characteristics observed in the degree-2 case continue and become more pronounced in degree-3 polynomials, as shown in Figure 4.6. Rational roots maintain a broad and relatively uniform distribution, which aligns with expectations given their derivation from factorizable expressions with integer coefficients. However, the irrational roots exhibit more clearly defined modes, with several localized peaks and gaps, suggesting underlying algebraic structures that govern their placements. Notably, the distribution of real roots — incorporating both rational and irrational values —

consistently occupies a middle ground between the other two, preserving both spread and concentration to a moderate extent.

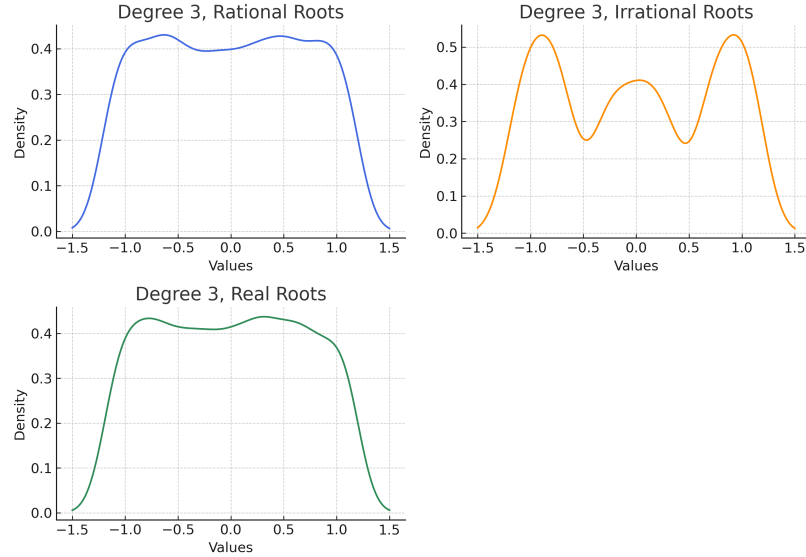


Figure 4.6. Kernel density plots of degree-3 polynomial roots.

This consistent pattern across degrees further confirms the stability and reproducibility of the symbolic root generation method and demonstrates its ability to capture root-type-specific complexity as polynomial degree increases.

As depicted in Figure 4.7, the complexity of root distributions becomes even more evident as the polynomial degree increases to four. The density curve corresponding to rational roots continues to exhibit a relatively flat and wide spread, indicating that these roots are sampled in a broadly uniform manner across the domain. In contrast, the distribution of irrational roots shows a significantly more intricate structure, marked by multiple local maxima and sharper peaks. This multimodality reflects the constrained algebraic processes used to generate such roots, which naturally cluster around certain symbolic expressions. The real root distribution, encompassing both rational and irrational components, again falls between these two extremes — maintaining a smoother appearance than the irrational distribution, yet more structured than the rational one. This pattern emphasizes how the symbolic generation framework adapts to increasing polynomial complexity, producing datasets that remain consistent in structure while

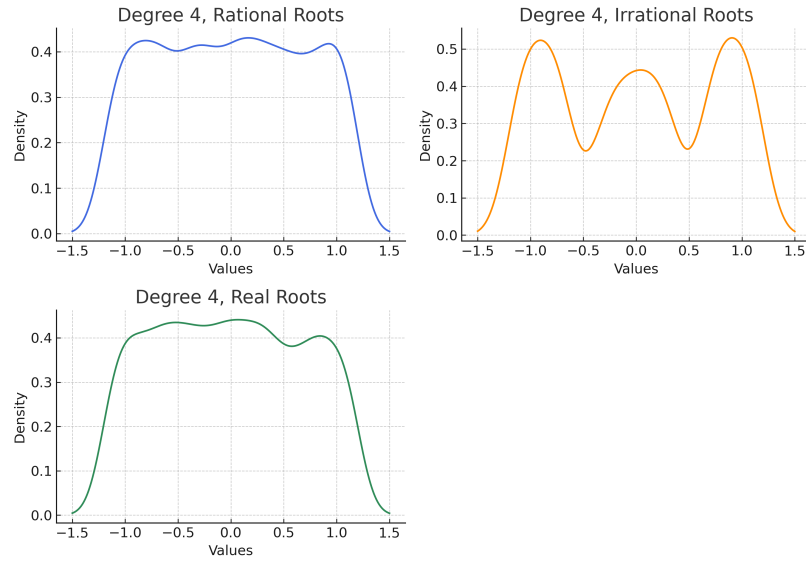


Figure 4.7. Kernel density plots of degree-4 polynomial roots.

encoding rich and meaningful distinctions tied to root types. The evolution of these patterns with degree provides strong evidence of the method’s robustness and scalability in capturing algebraically relevant features across different classes of polynomials.

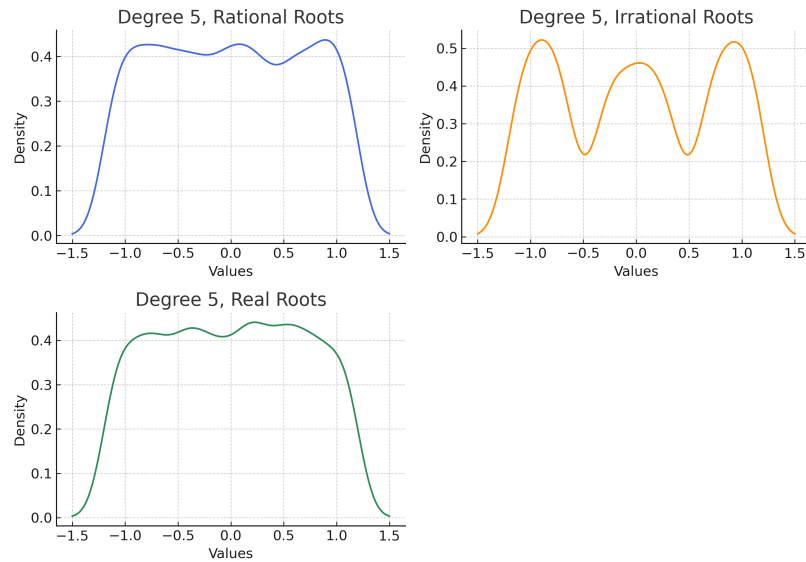


Figure 4.8. Kernel density plots of degree-5 polynomial roots.

Finally, in degree-5 polynomials (Figure 4.8), the distinctions among root types are still preserved. Despite greater variability, symbolic generation continues to yield root sets that are bounded, diverse, and structurally informative. These results confirm that the proposed dataset construction framework maintains both algebraic precision and distributional control as polynomial degree increases.

In previous examples (Figures 4.5, 4.6, 4.7 and 4.8), roots were sampled from the interval $[-1, 1]$, resulting in relatively concentrated coefficient distributions.

To investigate how the root interval affects the distribution of polynomial coefficients, we also constructed datasets where roots were sampled from a wider range, specifically $[-5, 5]$. The following figures illustrate the resulting coefficient distributions for degree-3 and degree-4 polynomials, each under different root types.

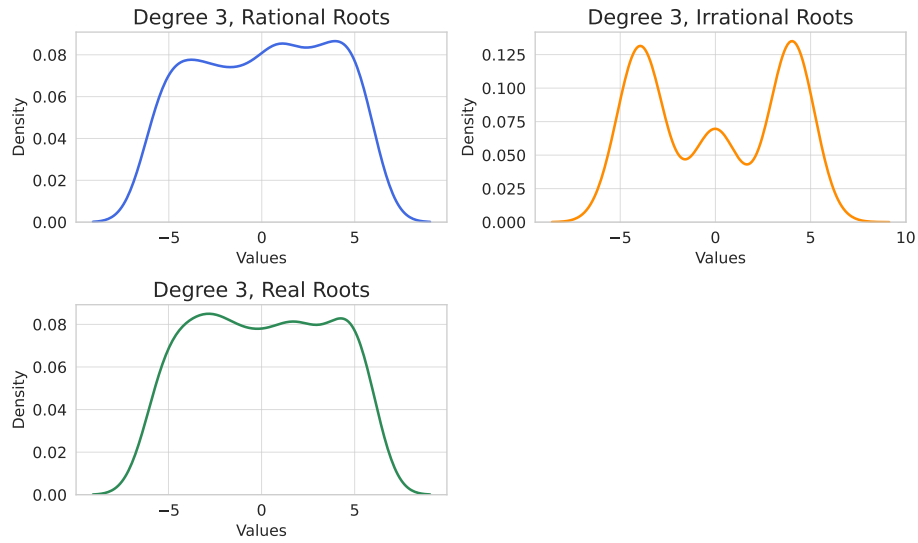


Figure 4.9. Root distribution for degree-3 polynomials with roots in $[-5, 5]$.

As shown in Figure 4.9, although the root types differ (rational, irrational, or general real), the overall coefficient distribution remains relatively structured. However, the wider root range leads to a more stretched support in the coefficient space, especially at the tails.

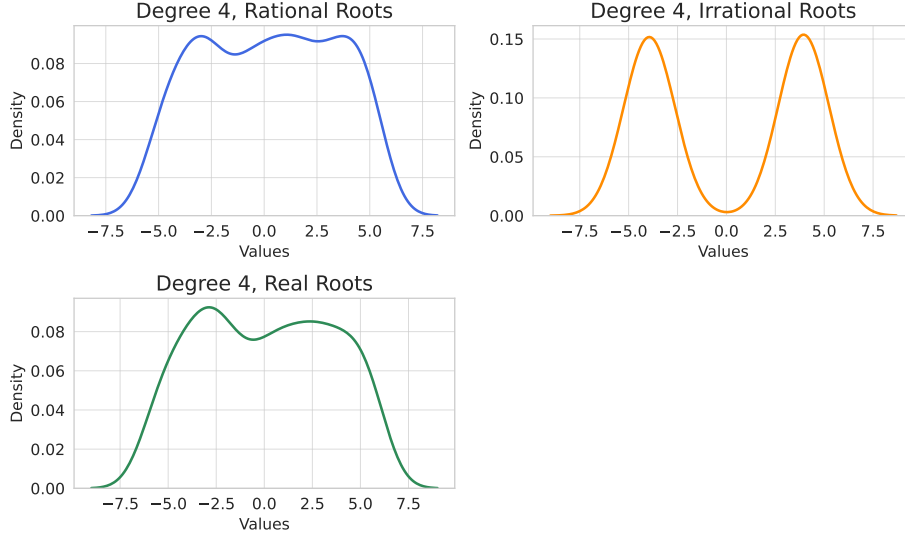


Figure 4.10. Root distribution for degree-4 polynomials with roots in $[-5, 5]$.

In contrast, the degree-4 case in Figure 4.10 reveals significantly more complex shapes in the coefficient distributions. The density becomes more multi-modal and less symmetric, especially when irrational roots are used. This suggests that both the degree and root type increase the complexity of the target distribution, making the learning task for G more difficult.

These visual analyses confirm that the symbolic root generation method yields clearly distinct distributions depending on root type. Although all roots lie within the same interval, their density profiles vary notably across rational, irrational, and mixed cases. As degree increases, distributions become more complex, demonstrating that symbolic choices lead to meaningful statistical variation in the generated datasets.

4.1.2. GAN Training Setup

4.1.2.1. Latent Space and Input Representation

The generator G receives as input a latent vector $r \in \mathbb{R}^{d+1}$ sampled uniformly from the interval $[-1, 1]$. This vector is augmented with a constant bias unit, resulting in an input vector of length $d + 2$. The latent space is designed to match the polynomial degree, ensuring that G has sufficient capacity to produce coefficients for degree- d

polynomials. Each element of r can be viewed as an abstract encoding of root-related features.

Overview of GAN Architecture.

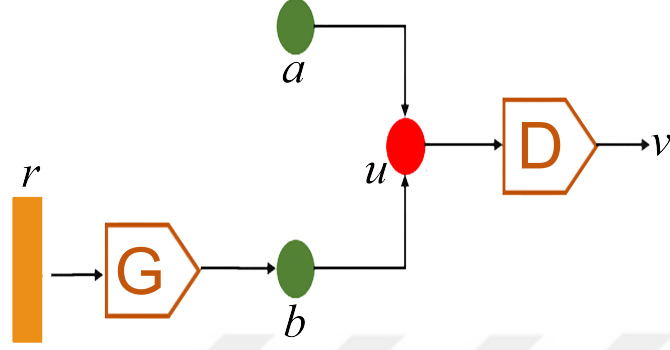


Figure 4.11. Schematic of our GAN architecture for generating real-rooted polynomials.

Notational summary of variables shown in the diagram:

- $r \in \mathbb{R}^{d+1}$ — Latent vector input to the generator.
- $b = G(r) \in \mathbb{R}^{d+1}$ — Generator output, coefficients of a fake polynomial.
- $a \in \mathbb{R}^{d+1}$ — Coefficient vector from the real symbolic dataset.
- $u \in \mathbb{R}^{d+1}$ — Input to the discriminator, randomly sampled from the shuffled union $\{a_i\} \cup \{b_j\}$.
- $v = D(u) \in [0, 1]$ — Discriminator output, indicating the probability of u being real.

4.1.2.2. Model Architecture

The architecture of G consists of two hidden layers with 5 neurons each, using the Exponential Linear Unit (ELU) as the activation function. The output of G is a real-valued vector $b = [b_0, b_1, \dots, b_d] \in \mathbb{R}^{d+1}$ that is interpreted as the coefficients of a degree- d polynomial, represented either in the monomial or Bernstein basis.

The discriminator D takes as input a coefficient vector of a degree- d polynomial and

returns a scalar output in the interval $[0, 1]$, indicating the probability that the input corresponds to a real polynomial. The discriminator also follows a two-layer architecture with 5 neurons in each hidden layer, using ELU activation followed by a final sigmoid activation function.

In this framework, G maps latent vectors r to polynomial coefficient vectors $b = G(r)$. The real dataset provides coefficient vectors $a = [a_0, a_1, \dots, a_d]$ corresponding to verified real-rooted polynomials. The input to the discriminator is the union of real and fake samples, denoted by $x = \{a : a \in \mathcal{R}\} \cup \{b : b = G(r)\}$, where each element of x is a $(d + 1)$ -dimensional coefficient vector. Before feeding into D , the combined input set x is randomly shuffled to ensure label information is not positionally biased. The discriminator learns to assign $D(x) \approx 1$ if x originates from the real dataset and $D(x) \approx 0$ if it is generated.

4.1.2.3. Generator and Discriminator Architectures

The architectures of the Generator (G) and Discriminator (D) used in our GAN setup are illustrated in Figures 4.12 and 4.13, respectively. Each model is built as a multilayer perceptron with fully connected layers. The input and output dimensions are adapted to the structure of the polynomial coefficient vectors.

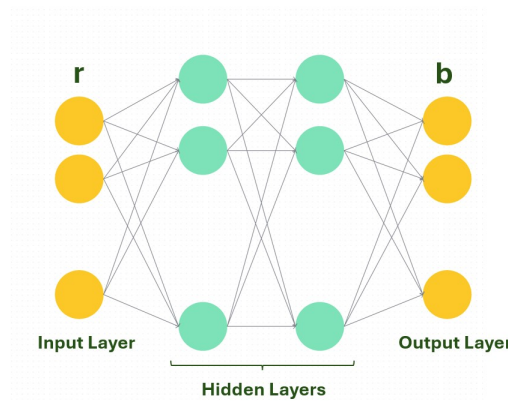


Figure 4.12. Generator (G) network architecture. The model takes a latent vector r as input and generates a candidate coefficient vector b .

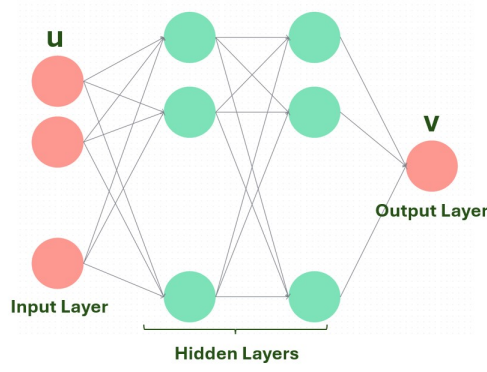


Figure 4.13. Discriminator (D) network architecture. The model receives a vector u and attempts to classify whether it is a real coefficient vector v or a fake one generated by G .

4.1.2.4. Loss Functions and Optimization

During training, each batch consists of half real and half fake examples. The real polynomials are sampled from a symbolic dataset of provably real-rooted polynomials, while the fake samples are generated via $G(r)$. The discriminator is trained using the binary cross-entropy loss:

$$\mathcal{J}_D(W_D, x, y) = -y \log D(W_D, x) - (1 - y) \log(1 - D(W_D, x)) \quad (4.18)$$

where $y = 1$ for real samples and $y = 0$ for generated samples. The discriminator seeks to maximize this objective, correctly classifying real and fake inputs.

The generator is trained with the goal of minimizing the loss:

$$\mathcal{J}_G(W_G, W_D, r) = \log(1 - D(W_D, G(W_G, r))) \quad (4.19)$$

This classic GAN formulation encourages G to produce outputs that D is likely to classify as real.

Training is performed using Stochastic Gradient Descent (SGD) with a learning rate of $\eta = 0.0002$. Gradients for both G and D are computed using automatic differentiation.

4.1.2.5. Training Procedure and Parameters

For each batch:

1. Real and fake polynomials are sampled and used to compute $\mathcal{J}_D$.
2. W_D is updated via gradient ascent.
3. New latent vectors r are sampled, fake outputs $G(r)$ are generated.
4. $\mathcal{J}_G$ is computed, and W_G is updated via gradient descent.

Each dataset is trained for a fixed number of epochs (typically 5 to 20), with a batch size of 100. Training is repeated separately for datasets with different degree, root type, and basis representation.

4.1.2.6. Motivation for Architectural Simplicity

The choice of this simple architecture was deliberate: the aim was not to construct a highly expressive generative model, but rather to evaluate how basic GAN behavior interacts with symbolic polynomial structures. This minimalistic setup allows us to isolate and observe the influence of root distribution, basis representation, and degree complexity on the success of GAN training. It also makes the experimental results interpretable and reproducible, setting a clear foundation for more complex architectures in future work.

4.1.2.7. Variational Autoencoder Architecture and Training

In addition to GANs, we also explored the use of Variational Autoencoders (VAEs) for generating real-rooted polynomials. A VAE is an unsupervised generative model that learns a compressed latent representation of input data and reconstructs it through a decoder network [36].

Architecture. Figure 4.14 shows the overall structure of the Variational Autoencoder used in this study. The encoder maps the input coefficient vector a to a latent vector z , and the decoder reconstructs the output coefficient vector b from z .

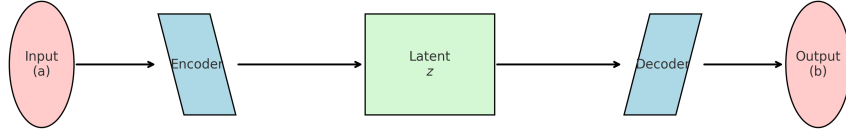


Figure 4.14. VAE structure.

The VAE model consists of two neural networks:

- The **encoder** network $q_\phi(z|x)$ takes an input polynomial represented by its coefficient vector $x = [a_0, \dots, a_d]$ and maps it to a latent vector $z \in \mathbb{R}^k$. The encoder outputs the mean μ and standard deviation σ of a multivariate Gaussian distribution in latent space.
- The **decoder** network $p_\theta(x|z)$ takes a latent vector z and attempts to reconstruct the input coefficient vector x .

The latent variable z is sampled using the reparameterization trick:

$$z = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I) \quad (4.20)$$

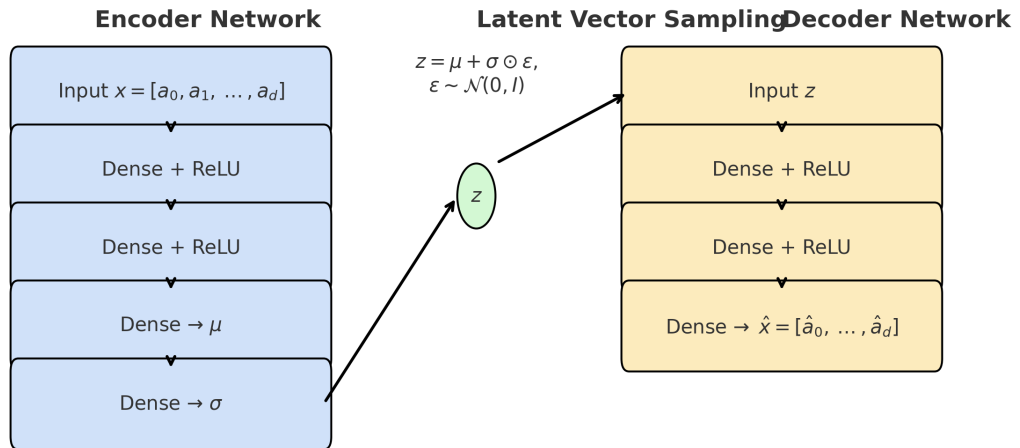


Figure 4.15. Encoder and decoder layers.

Figure 4.15 illustrates the internal architecture of the encoder and decoder networks. The encoder produces the mean and standard deviation vectors for the latent variable

using fully connected layers with ReLU activation. The latent vector z is sampled using the reparameterization trick and passed to the decoder, which reconstructs the polynomial coefficient vector.

Loss Function. The training objective is to maximize the Evidence Lower Bound (ELBO), which encourages the model to reconstruct the input accurately while keeping the learned latent distribution close to the prior:

$$\mathcal{L}_{\text{ELBO}}(x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \text{KL}(q_\phi(z|x) \parallel \mathcal{N}(0, I)) \quad (4.21)$$

This loss is minimized using backpropagation. The first term encourages accurate reconstruction of the input coefficients, while the second term ensures the latent space is smooth and regularized.

Implementation Details. In our experiments, we implemented VAE models for polynomials of degree $d = 2, 3, 4$, using monomial basis and a coefficient vector $x \in \mathbb{R}^{d+1}$. The latent space dimension was fixed at $k = 3$ across all experiments. Both encoder and decoder networks were implemented as simple feed-forward MLPs with ReLU activations and batch normalization. The models were trained using the Adam optimizer for 1000 epochs with a learning rate of 0.001 and batch size of 64.

4.2. GAN with Pretrained Discriminator

In addition to the direct generation method introduced earlier, we explored a second approach to enhance the quality of real-rooted polynomial generation by integrating a discriminative model into the training process. This method involves a binary classifier trained to distinguish between real-rooted and non-real-rooted polynomials, which is then used as a Discriminator (D) in a custom GAN setup.

The methodology consists of the following steps:

- **Dataset Preparation:** We first created a dataset containing both real-rooted and non-real-rooted polynomials. Each polynomial is represented by a pair of vectors $(\mathbf{a}, \mathbf{c})$ encoding its coefficient structure and constraint type.
- **Binary Classification:** A multi-layer perceptron (MLP) model was trained to perform binary classification over this dataset. After evaluating various architec-

tures, the best-performing model—based on classification accuracy and generalization—was selected to serve as the Discriminator (D).

- **GAN Training Setup:** Unlike traditional GANs where both real and generated data are used to train D , we froze the trained D and only used it to guide the training of the Generator (G). The training objective of G was to produce polynomial representations ($\mathbf{a}, \mathbf{c}$) such that D classifies them as real-rooted. The output of $D(G(z))$ was used as feedback for optimizing G .
- **Generator Utilization:** After the training process, the final Generator was used as a real-rooted polynomial generator, similar to the previous approach.

This strategy allowed us to leverage a learned discriminator model as an algebraic filter, effectively embedding real-rootedness verification into the generative process, without requiring explicit mathematical constraints like the discriminant. Compared to the earlier method, this framework provides more flexibility and modularity by decoupling constraint learning from data generation.

4.2.1. Dataset Construction

For each polynomial degree $d \in \{2, 3, 4, 5\}$, a binary classification dataset was constructed to distinguish between real-rooted polynomials (RRPs) and polynomials with at least one pair of nonreal complex roots (non-RRPs).

Root Sampling Strategy. Polynomials were constructed by sampling their roots directly, rather than drawing coefficient vectors randomly. This method provides precise control over root structure and avoids numerical issues caused by unstable or ill-conditioned polynomials.

- **Real roots** were sampled uniformly from the interval $[-1, 1]$. For an RRP, the roots are denoted as $x_1, x_2, \dots, x_d \in \mathbb{R}$.
- **Complex roots** were generated in conjugate pairs:

$$z_i = x_i + iy_i, \quad \bar{z}_i = x_i - iy_i, \quad \text{where } x_i, y_i \in [-1, 1], \quad y_i \neq 0. \quad (4.22)$$

This ensures that the resulting polynomial remains real while exhibiting nonreal root behavior.

Each polynomial was then constructed by multiplying the corresponding linear and quadratic factors:

$$P(x) = \prod_{i=1}^k (x - x_i) \cdot \prod_{j=1}^m (x^2 - 2x_jx + x_j^2 + y_j^2), \quad (4.23)$$

where the first term corresponds to real roots, and the second term corresponds to conjugate complex pairs.

Coefficient Normalization and Labeling. Let $\mathbf{a} = [a_0, a_1, \dots, a_d]$ denote the coefficient vector of a real-rooted polynomial (RRP), and let $\mathbf{c} = [c_0, c_1, \dots, c_d]$ denote that of a non-RRP. To ensure numerical stability and training consistency, all coefficient vectors were normalized using the L2 norm:

$$\tilde{\mathbf{a}} = \frac{\mathbf{a}}{\|\mathbf{a}\|_2}, \quad \tilde{\mathbf{c}} = \frac{\mathbf{c}}{\|\mathbf{c}\|_2}, \quad \text{where } \|\mathbf{v}\|_2 = \sqrt{\sum_{i=0}^d v_i^2}. \quad (4.24)$$

This operation preserves the root structure of the polynomial, as it merely scales the entire polynomial by a constant factor.

Each polynomial was labeled as follows:

- Label 1 for RRP (all roots real),
- Label 0 for non-RRPs (at least two complex roots).

Dataset Sizes and Splitting. For each degree d , a total of $2n$ polynomials were generated — n RRP and n non-RRPs. This combined dataset was then randomly shuffled and split into training, validation, and test sets in an 80%–10%–10% ratio, maintaining class balance across all subsets.

4.2.2. Binary Classification Training Setup

Before integrating the Discriminator into the GAN framework, we first trained it as an independent binary classifier to distinguish between real-rooted polynomials (RRP) and non-real-rooted polynomials (non-RRP). The training procedure is outlined as follows:

Input Representation. Each polynomial sample is represented using a pair of coefficient vectors $(\mathbf{a}, \mathbf{c})$, where $\mathbf{a} = [a_0, a_1, \dots, a_d]$ are the monomial coefficients and $\mathbf{c} = [c_0, c_1, \dots, c_d]$ are the corresponding Bernstein basis coefficients.

Table 4.2. Dataset size and splits for each degree d . Each subset is class-balanced with 50% RRP and 50% non-RRP samples.

Degree d	Total	Train	Validation	Test
2	2000	1600	200	200
3	3000	2400	300	300
4	4000	3200	400	400
5	6000	4800	600	600

Dataset Composition. The dataset consists of labeled instances: real-rooted polynomials (label 1) and non-real-rooted polynomials (label 0). We used the symbolic datasets generated in earlier stages of this study, covering various polynomial structures and root types.

Network Architecture. A standard multilayer perceptron (MLP) architecture was employed. Several configurations were evaluated. The best performing one had the following hidden layer sizes: 128 -- 64 -- 32, followed by a sigmoid-activated output layer for binary classification.

Activation Functions. ReLU activations were used in hidden layers due to their training efficiency, and a sigmoid activation was used in the output layer to produce a probability score.

Loss Function and Optimization. The model was trained using binary cross-entropy loss. Optimization was performed using the Adam optimizer with standard parameters. Early stopping was applied based on validation accuracy to avoid overfitting.

Evaluation Metrics. Accuracy, precision, recall, and F1-score were computed on the validation set. The selected model achieved the highest performance across all metrics and was subsequently fixed as the Discriminator (D) in the GAN training.

This initial training phase allowed us to isolate and validate a highly effective Discriminator before using it in a generative adversarial setting.

4.2.2.1. Objective

The objective of this stage is to develop a neural network model that can reliably distinguish between real-rooted and non-real-rooted polynomials based solely on their coefficient vectors. This classifier is not only evaluated as a stand-alone binary classification model but also serves as a critical component in the subsequent GAN framework, where it is integrated as the discriminator.

By accurately learning the algebraic characteristics of real-rootedness from data, the discriminator helps guide the generator towards producing polynomials that conform to the real-rooted constraint. Thus, the goal is twofold: (i) achieving high classification accuracy on the binary task, and (ii) ensuring that the discriminator provides meaningful feedback for generative training.

4.2.2.2. Method

To train a binary classifier that distinguishes real-rooted polynomials (RRP) from non-RRP instances, we constructed a dataset composed of coefficient vectors $\mathbf{a} = [a_0, a_1, \dots, a_d]$ and the corresponding binary labels: $y = 1$ for real-rooted and $y = 0$ otherwise.

The dataset was designed to reflect the diversity of root structures across polynomials of a given degree. Real-rooted polynomials were generated by sampling real roots $x_1, \dots, x_d \in \mathbb{R}$ and computing the corresponding coefficient vector using the canonical form:

$$f(x) = (x - x_1)(x - x_2) \cdots (x - x_d) \quad (4.25)$$

Non-real-rooted polynomials were generated either by introducing complex conjugate root pairs or by perturbing real roots to produce complex roots, ensuring a clear distinction between the two classes.

After generating the labeled dataset, all coefficient vectors were normalized to lie within a bounded range to stabilize training. The dataset was split into training and validation sets with a standard 80/20 ratio.

The model—a feedforward neural network—was trained using binary cross-entropy loss. Optimization was performed using the Adam optimizer with a learning rate of 10^{-3} . Early stopping based on validation loss was employed to prevent overfitting.

The final model was selected based on its highest validation accuracy, which was later integrated into the proposed GAN architecture as the discriminator.

4.2.2.3. Model Architecture

The binary classification model employed a multilayer perceptron (MLP) architecture consisting of three fully connected hidden layers with progressively decreasing sizes: 128, 64, and 32 units, respectively. The model was designed to classify polynomial representations (via their coefficient vectors) as either real-rooted or not.

Each hidden layer uses the ReLU (Rectified Linear Unit) activation function to introduce nonlinearity and facilitate effective gradient propagation. The final output layer is a single neuron with a sigmoid activation, which maps the output to the interval $[0, 1]$, representing the probability that a given input corresponds to a real-rooted polynomial.

The architecture can be summarized as follows:

- **Input Layer:** $(d + 1)$ -dimensional coefficient vector $\mathbf{a} = [a_0, a_1, \dots, a_d]$
- **Hidden Layer 1:** Dense layer with 128 units, ReLU activation
- **Hidden Layer 2:** Dense layer with 64 units, ReLU activation
- **Hidden Layer 3:** Dense layer with 32 units, ReLU activation
- **Output Layer:** Dense layer with 1 unit, Sigmoid activation

The model was trained using the binary cross-entropy loss and optimized with the Adam optimizer.

4.2.2.4. Training and Evaluation

The binary classification model was trained using a labeled dataset containing both real-rooted and non-real-rooted polynomials, represented by their coefficient vectors. We employed a supervised learning approach with a standard binary cross-entropy loss function. The input vectors were normalized before training to improve convergence and numerical stability. The model was trained using the Adam optimizer with a fixed learning rate and batch size.

To assess the model's effectiveness, we monitored the classification accuracy on both

training and validation sets during training. The final architecture was selected based on its validation performance. Metrics such as accuracy, precision, recall, and loss curves were recorded to evaluate model quality.

4.2.3. Performance Metrics and Results of Binary Classification

To evaluate the classification performance, we used standard metrics:

- **Accuracy:** The proportion of correctly classified polynomials over the total number of samples.
- **Precision:** The proportion of polynomials classified as real-rooted that are actually real-rooted.
- **Recall (True Positive Rate):** The proportion of real-rooted polynomials that are correctly classified.
- **F1-Score:** The harmonic mean of precision and recall.
- **Validation Loss:** Binary cross-entropy loss evaluated on the validation set.

In this setup, training and validation accuracy curves closely follow each other, indicating stable training.

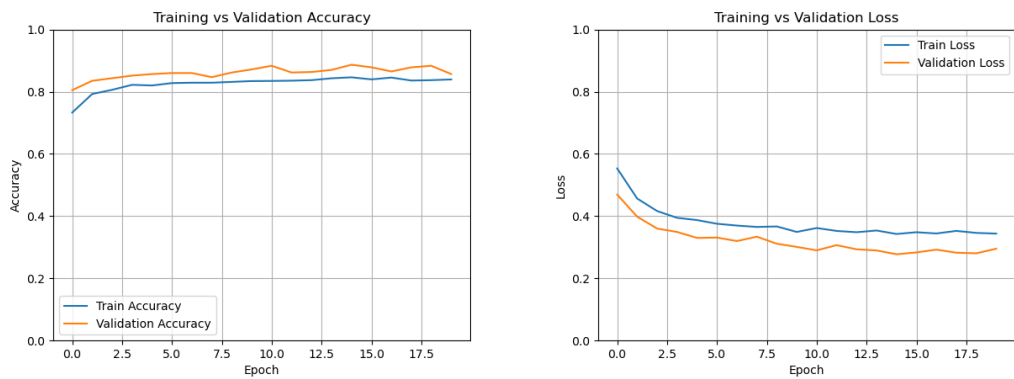


Figure 4.16. Training and validation accuracy (left) and loss (right) for degree 3 polynomials using the 64–32 architecture with dropout rate 0.2.

Loss curves steadily decrease, confirming convergence of the model.

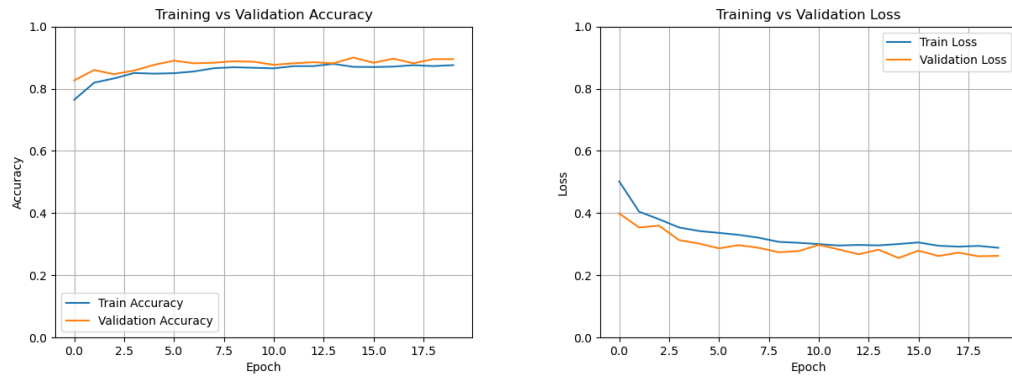


Figure 4.17. Training and validation accuracy (left) and loss (right) for degree 4 polynomials using the 128–64–32 architecture with dropout rate 0.1.

This configuration achieves consistent 89% validation accuracy while maintaining a decreasing loss trend. The accuracy plateau suggests early convergence.

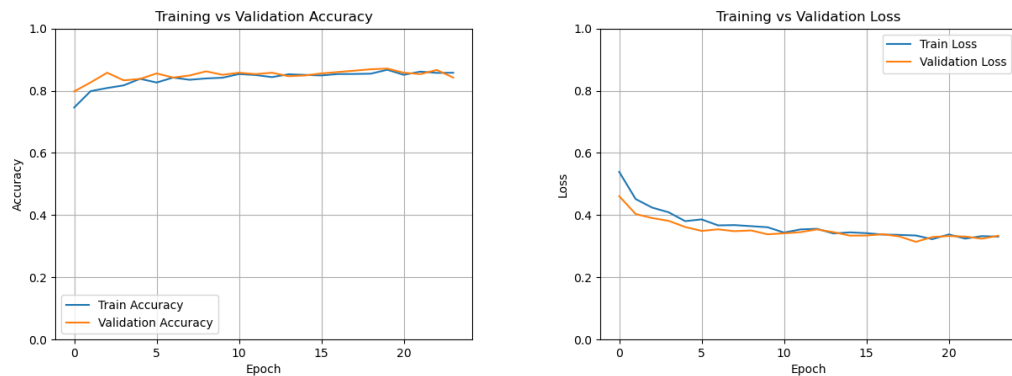


Figure 4.18. Training and validation accuracy (left) and loss (right) for degree 5 polynomials using the 128–64–32–16 architecture with dropout rate 0.1.

Despite the increased model complexity, validation accuracy remains around 86%, indicating that deeper architectures did not lead to performance improvement for this degree.

4.2.3.1. Results

For polynomials of degree 3, multiple architectures were evaluated. The accuracy values obtained for 128–64–32 (with dropout 0.1 and 0.2) and 128–64–32–16 (with dropout 0.1) were all identical, achieving 88%. Although 64–32 was slightly simpler, it yielded 1% lower accuracy. Thus, 128–64–32 was selected as a robust yet efficient architecture.

Table 4.3. Discriminator performance for degree 3.

Layers	Dropout	Accuracy (%)
128-64-32	0.1	88
128-64-32	0.2	88
128-64-32-16	0.1	88
64-32	0.1	87

In the case of degree 4, all evaluated architectures—both shallow and deep—achieved the same accuracy of 89%. Given its consistency and compatibility with other degrees, the 128–64–32 architecture was again favored.

Table 4.4. Discriminator performance for degree 4.

Layers	Dropout	Accuracy (%)
64-32	0.1	89
64-32-16	0.1	89
128-64-32	0.1	89
128-64-32	0.2	89

For degree 5 polynomials, accuracy remained unchanged (86%) across all architectures.

Despite the similar performance, the 128–64–32 model was chosen due to its optimal depth and alignment with previous results.

Table 4.5. Discriminator performance for degree 5.

Layers	Dropout	Accuracy (%)
128-64-32	0.1	86
128-64-32-16	0.1	86
128-64-32-16	0.2	86
64-32-16	0.1	86
64-32-16	0.2	86

Based on the accuracy performance across different degrees, the 128–64–32 architecture with 0.1 dropout was selected as the discriminator structure in the final GAN setup.

4.2.4. GAN Training Setup

The adversarial training process is divided into two consecutive phases, during which the Discriminator D and Generator G are optimized with different criteria and data sources. The purpose of this strategy is to leverage supervised learning for D while allowing G to learn from D 's responses.

Phase 1: Simultaneous Training with Fixed Roles.

Initially, both D and G are trained in parallel but using separate data and objectives. The Discriminator D is trained in a supervised fashion using the labeled real dataset (x, y) , where $x \in \mathbb{R}^{d+1}$ represents a normalized polynomial coefficient vector and $y \in \{0, 1\}$ denotes the label (RRP or non-RRP). Standard binary cross-entropy (BCE) loss is used to update the weights of D .

Meanwhile, the Generator G receives random latent vectors $r \in \mathbb{R}^{d+1}$ as input and produces synthetic coefficient vectors $G(r) \in \mathbb{R}^{d+1}$. These are passed through the Discriminator to obtain $D(G(r))$, and the Generator is trained to maximize this output (i.e., to make $D(G(r)) \approx 1$). Gradient updates for G are performed using the BCE loss

with positive targets.

During this phase, early stopping is monitored only for the Discriminator using its validation accuracy. Once D reaches optimal performance on the validation set, its weights are frozen and preserved for the remainder of the training process.

Phase 2: Generator Optimization with Frozen Discriminator.

After D is frozen, the Generator G continues to be trained independently using feedback from the fixed D . The objective remains the same: to increase $D(G(r))$, thereby making the generated polynomials indistinguishable from real ones in the eyes of the Discriminator.

An early stopping mechanism is now activated for the Generator based on the behavior of a performance metric we call $accuracy_G$:

$$accuracy_G = \frac{1}{N} \sum_{i=1}^N \mathbb{I}[D(G(r_i)) > 0.5], \quad (4.26)$$

where $\mathbb{I}[\cdot]$ is the indicator function and r_i are random latent vectors sampled from a uniform distribution.

Activation Function Selection in Generator and Discriminator. In the design of GAN-based models for generating polynomial coefficient vectors, the choice of activation functions plays a crucial role in both training stability and output behavior. In our implementation, different activation functions are used for the hidden and output layers of the Generator (G) and Discriminator (D), motivated by their mathematical properties and empirical effectiveness.

Hidden Layers. For both G and D , the hidden layers utilize the Rectified Linear Unit (ReLU) activation:

$$\text{ReLU}(x) = \max(0, x) \quad (4.27)$$

ReLU is widely used in deep learning due to its computational efficiency and its ability to mitigate vanishing gradient issues. In our setting, it provides sparsity and a strong nonlinear transformation without saturating, which helps both the generator and discriminator learn more effectively from high-dimensional data.

Output Layers.

- **Generator Output Layer:** The final layer of the generator uses the tanh activation function:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \tanh(x) \in (-1, 1) \quad (4.28)$$

Since we aim to generate polynomial coefficients that are symmetrically distributed around zero (i.e., without favoring positive or negative values), tanh is the most natural choice. It ensures that the output values are bounded in a symmetric range, which leads to balanced positive and negative coefficients.

- **Discriminator Output Layer:** The discriminator ends with the sigmoid activation:

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad \sigma(x) \in (0, 1) \quad (4.29)$$

The sigmoid function is suitable for binary classification tasks. It maps the output to a probability space, which allows the discriminator to effectively distinguish between real and generated polynomials.

Comparison of Activation Functions. The following table summarizes the activation functions used in our architecture, along with their key properties and the rationale behind their selection:

Table 4.6. Activation functions used in Generator and Discriminator networks.

Component	Layer	Activation	Range	Motivation
Generator (G)	Hidden	ReLU	$[0, \infty)$	Efficient nonlinearity that avoids saturation and accelerates training.
Generator (G)	Output	tanh	$[-1, 1]$	Encourages balanced positive and negative coefficient generation; suitable for symmetric distributions.
Discriminator (D)	Hidden	ReLU	$[0, \infty)$	Standard activation function for deep networks due to computational efficiency.
Discriminator (D)	Output	Sigmoid	$[0, 1]$	Maps outputs to probabilities for binary classification (real vs. fake).

Conclusion. The combination of tanh in the generator output and sigmoid in the discriminator output aligns with the standard GAN setup, but our use of tanh is particularly important for preserving the symmetry of polynomial coefficients. This activation strategy not only improves the realism of generated samples but also ensures that the GAN learns a more balanced and expressive latent space.

Early Stopping Criteria and Evaluation Frequency. During adversarial training, the Discriminator (D) is evaluated at the end of each epoch using a validation set. This ensures that the model is not overfitted to minibatch fluctuations and that the generalization performance is accurately measured. Once D achieves its best validation accuracy, its weights are frozen and retained for the second training phase.

On the other hand, the Generator (G) is evaluated more frequently — after every minibatch update — during the second phase. This design choice is motivated by the fact that G relies solely on feedback from the already-trained and fixed D . Since G is not trained on labeled data and does not have a separate validation set, its performance must be monitored using an internal metric such as *accuracyG*, which measures how often $D(G(z)) > 0.5$ for generated samples.

This frequent evaluation allows fine-grained monitoring of G 's progress and enables early stopping as soon as the improvement plateaus or undesirable behaviors emerge. Hence, evaluation frequency is asymmetrical between D and G : D is validated once per epoch (supervised), while G is monitored every minibatch (unsupervised with a fixed critic).

4.2.4.1. Results

Figure 4.19 shows a relatively stable training performance for degree 3 polynomials. The generator quickly adapts to produce valid real-rooted polynomials, and the discriminator maintains a consistently high accuracy.

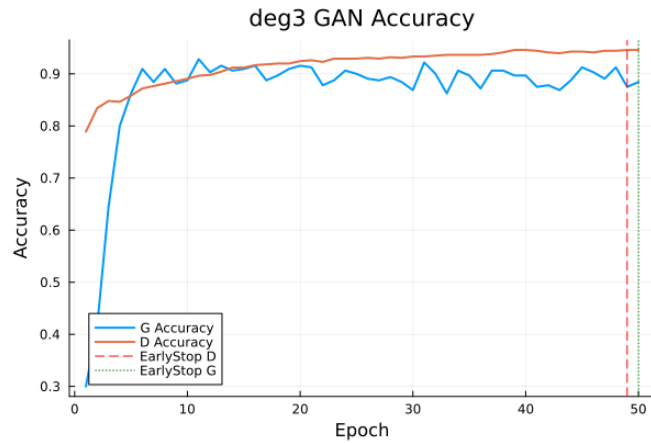


Figure 4.19. GAN training results for degree 3. Discriminator accuracy: 87.81%, Generator accuracy: 89.22%.

Figure 4.20 illustrates an almost ideal scenario where the generator achieves perfect accuracy, indicating that it successfully learned the real-rooted polynomial structure, as confirmed by the classifier (D).

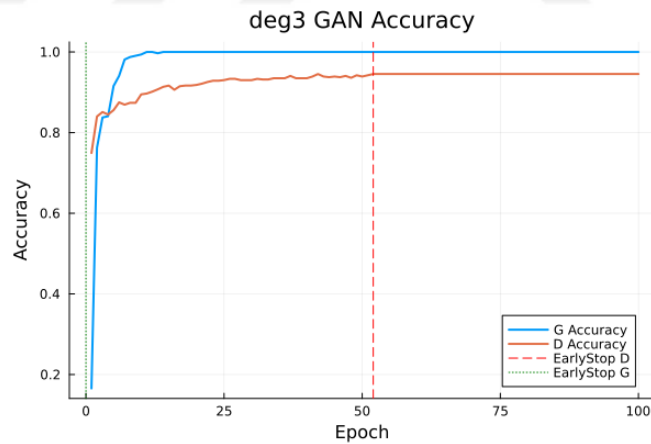


Figure 4.20. GAN training results for degree 3 (second run). Discriminator accuracy: 90.31%, Generator accuracy: 100.00%.

Figure 4.21 shows that while the discriminator performs well, the generator struggles to match the required structure. This could be due to increased complexity in degree 4 polynomial structures or limitations in the generator's expressivity.

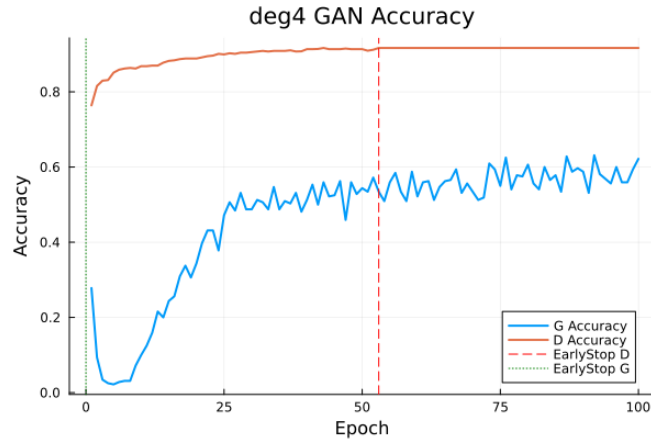


Figure 4.21. GAN training results for degree 4. Discriminator accuracy: 90.05%, Generator accuracy: 61.72%.

In Figure 4.22, although the discriminator maintains high classification accuracy, the generator fails to produce valid real-rooted polynomials. Its accuracy remains near zero throughout the training.

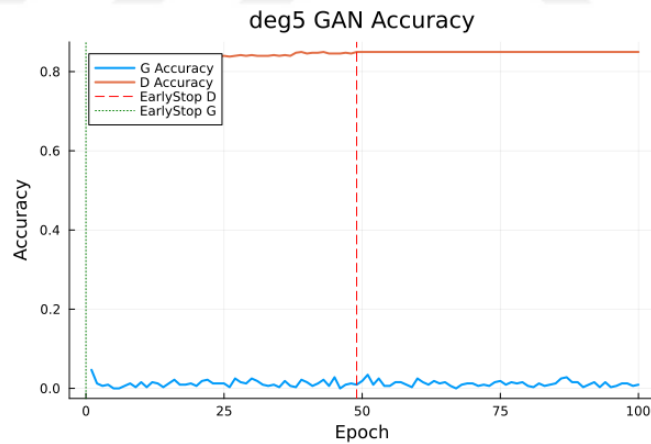


Figure 4.22. GAN training results for degree 5. Discriminator accuracy: 91.41%, Generator accuracy: 1.25%.

This result confirms the increasing difficulty in generating real-rooted polynomials as the degree increases, especially without specific structural constraints in the loss function.

5. EXPERIMENTAL RESULTS

This chapter presents the outcomes of the experimental evaluation of the proposed generative model. We report the training setup, dataset configurations, evaluation metrics, and key observations regarding model performance.

5.1. Dataset Variants and Model Configurations

5.1.1. Dataset Variants and Model Configurations for GAN

A total of 56 symbolic datasets were constructed for experimentation. Each dataset contains 1,000 training and 200 testing polynomials, and is characterized by a unique combination of:

- Degree: $d \in \{2, 3, 4, 5\}$,
- Root range: $[-1, 1]$, $[-2, 2]$, or $[-5, 5]$,
- Root type: rational only, irrational only, or mixed real,
- Basis: monomial or Bernstein.

For each of these 56 configurations, a separate GAN model was trained. The architecture of the generator and discriminator consisted of two hidden layers, and experiments were repeated using different hidden layer sizes: $[8, 8]$, $[16, 16]$, $[32, 32]$, and $[64, 64]$.

Each model was trained for 10, 20, and 50 epochs to investigate how learning performance evolves over training duration.

5.1.2. Dataset Variants and Model Configurations for VAE

For VAE experiments, the input data consisted of real-rooted polynomials in monomial basis with coefficients normalized to lie in the $[-1, 1]$ range. Three datasets were prepared for each degree: polynomial sets with rational roots, irrational roots, and general real roots. These datasets were merged into a unified training and test set per degree. Degree-2, 3, and 4 polynomials were tested, with input dimensions 3, 4, and 5 respectively. Each model used a latent space of dimension 2 or 3 depending on the

experiment.

5.2. Evaluation Metrics

5.2.1. Evaluation Metrics for GAN

During training, we tracked:

- Discriminator accuracy (D Accuracy),
- Generator accuracy (G Accuracy),
- Precision, Recall, and F1-score of discriminator predictions.

While the latter three metrics were computed for completeness, their interpretation is limited due to the generative nature of the task. Instead, discriminator accuracy and qualitative inspection of generated outputs were more informative.

5.2.1.1. Discriminator Accuracy

To evaluate the learning progress of the discriminator D , we compute its classification accuracy at each epoch. Given a batch of input vectors $\mathbf{u}_i \in \mathbb{R}^{d+1}$ (either real or generated coefficient vectors) and their corresponding binary labels $y_i \in \{0, 1\}$, the output of the discriminator is:

$$v_i = D(\mathbf{u}_i), \quad v_i \in [0, 1] \quad (5.1)$$

Predictions are made by applying a 0.5 threshold:

$$\hat{y}_i = \begin{cases} 1 & \text{if } v_i \geq 0.5 \\ 0 & \text{otherwise} \end{cases} \quad (5.2)$$

The discriminator accuracy is then calculated as the proportion of correct predictions:

$$\text{Accuracy}_D = \frac{1}{m} \sum_{i=1}^m 1_{\{\hat{y}_i = y_i\}} \quad (5.3)$$

This metric captures the ability of D to distinguish real samples from fake ones. In the context of GANs, a high D accuracy may indicate that G is underperforming, while a D accuracy around 0.5 may suggest that G is successfully confusing the discriminator.

5.2.1.2. Generator Accuracy

To evaluate the effectiveness of the generator G , we introduce a metric that directly measures the proportion of generated polynomials that are real-rooted.

Let $G(\mathbf{r}_i) = \mathbf{b}^{(i)} = [b_0^{(i)}, b_1^{(i)}, \dots, b_d^{(i)}]$ be the coefficient vector produced by G from latent input $\mathbf{z}_i$. We define the polynomial:

$$p^{(i)}(x) = b_0^{(i)} + b_1^{(i)}x + \dots + b_d^{(i)}x^d \quad (5.4)$$

For each $p^{(i)}(x)$, we compute its real roots using symbolic root-finding via sympy. The generator is considered successful on that instance if all d roots are real. That is, if:

$$\text{length}(\text{real_roots}(p^{(i)})) = d \quad (5.5)$$

Then the generator accuracy over a batch of m samples is defined as:

$$\text{Accuracy}_G = \frac{1}{m} \sum_{i=1}^m 1_{\{p^{(i)} \in \mathcal{R}\}} \quad (5.6)$$

where $\mathcal{R}$ denotes the set of real-rooted polynomials, and $1_{\{\cdot\}}$ is the indicator function.

This metric provides a direct measure of how well G is learning the structural constraints of the target distribution. Note that this computation is symbolic and exact, and only applicable during evaluation, not during training.

5.2.1.3. Additional Evaluation Metrics

To further evaluate the performance of the discriminator D , we computed classical classification metrics based on true/false positives and negatives. For each prediction, we compared $D(\mathbf{u}) \geq 0.5$ to the ground-truth label $y \in \{0, 1\}$, and assigned:

- TP (True Positive): predicted 1, and $y = 1$

- TN (True Negative): predicted 0, and $y = 0$
- FP (False Positive): predicted 1, but $y = 0$
- FN (False Negative): predicted 0, but $y = 1$

From these counts, we computed the following:

$$\text{Accuracy}_D = \frac{TP + TN}{TP + TN + FP + FN} \quad (5.7)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (5.8)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (5.9)$$

$$\text{F1-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (5.10)$$

5.2.2. Evaluation Metrics for VAE

Unlike GANs, where generator performance is indirectly inferred from the discriminator response, VAE allows direct evaluation of the decoder outputs. After training, random vectors were sampled from the latent space and passed through the decoder to generate new polynomials. A polynomial is considered successful if all its roots are real.

Thus, the evaluation metric is defined as the percentage of decoded polynomials that are real-rooted among all generated polynomials. This success rate was computed over 60,000 decoded samples. To account for randomness, the metric was averaged across 10 trials. Standard deviation and success range were also reported.

5.3. Experimental Results with GAN

5.3.1. Interpreting Evaluation Metrics in GAN Training

Although classical classification metrics such as Precision, Recall, Accuracy, and F1-score are widely used in evaluating discriminative models, their interpretation within

the Generative Adversarial Network (GAN) framework differs substantially from traditional settings. In this study, these metrics were primarily monitored for the discriminator D to assess whether a meaningful adversarial interaction emerged between D and the generator G .

Expected Metric Behavior When G Is Learning Successfully

In a well-functioning GAN, the generator G gradually learns to produce data that closely mimics the real data distribution. As G improves, the discriminator D becomes increasingly unable to distinguish between real and generated samples. This loss of discriminative ability leads to the following expected behavior in classical metrics:

- **Precision** ($\frac{TP}{TP+FP}$) tends to decrease, as D starts classifying fake samples as real (increased FP) while its true positive predictions decline.
- **Recall** ($\frac{TP}{TP+FN}$) also decreases, because D may misclassify more real samples as fake (increased FN).
- **F1-score**, the harmonic mean of Precision and Recall, also decreases.
- **Accuracy** approaches 0.5, indicating that D is essentially guessing.

This counterintuitive decline in performance is a positive signal in the GAN setting: it indicates that G is generating realistic samples that confuse D . Therefore, when these metrics collectively decrease and D 's accuracy converges to 0.5, it can signify effective training of G .

When the Metrics Suggest D Is Performing Well

Conversely, if D maintains high accuracy, precision, recall, and F1-score throughout training, this likely means that G has failed to generate convincing outputs. D is not challenged and can easily separate real and fake samples. Such a scenario reflects a lack of adversarial tension, meaning G is not improving and GAN training is ineffective.

Detecting Lack of Interaction Between G and D

Sometimes, the computed metrics conflict with qualitative assessments of generated samples. For instance:

- If the metrics (e.g., precision and recall) suggest that G is improving, but visual or symbolic inspection reveals poor-quality outputs, then G and D may not be influencing each other effectively. G could be evolving independently of D .
- Conversely, if D appears highly accurate and precise, but G is nonetheless generating high-quality outputs, this again points to a decoupling of their dynamics. The adversarial loop may have broken, and G may be optimizing in a disconnected way.

Remarks on Metric Trends

In the GAN setting, traditional classification metrics must be interpreted with caution. They are not absolute indicators of model success but tools to diagnose the interaction between G and D . Their usefulness depends on context, and they should always be analyzed in conjunction with qualitative evaluations of generated outputs.

5.3.2. Preliminary Success: GANs for Degree-2 Real-Rooted Polynomials

Before the symbolic dataset experiments presented in this thesis, a focused study was conducted to investigate whether a simple GAN model could learn to generate degree-2 real-rooted polynomials (RRPs). This experiment served both as a proof-of-concept and as motivation for the broader symbolic generation task explored in the main body of the thesis.

Motivation and Setup

The motivation was to assess whether a neural network—specifically a GAN—could internalize the algebraic structure associated with real-rootedness, even in the simplest case of quadratic polynomials. Degree-2 polynomials offer a unique opportunity be-

cause their discriminant $\Delta = b^2 - 4ac$ provides a direct, closed-form test of whether their roots are real. Furthermore, this discriminant can be differentiated and incorporated directly into a loss function, which is not easily possible for higher-degree cases. To build the dataset, two integer roots $x_1, x_2 \in \{1, 2, \dots, 32\}$ were selected randomly and used to form polynomials of the form:

$$p(x) = (x - x_1)(x - x_2) = x^2 - (x_1 + x_2)x + x_1x_2. \quad (5.11)$$

expanded into coefficient vectors $[b_0, b_1, b_2]$ with $b_2 = 1$, $b_1 = -(x_1 + x_2)$, and $b_0 = x_1x_2$, then normalized to lie in $[-1, 1]$ for compatibility with the generator's output range.

The training set included 1000 such polynomials, and the test set included 200.

Discriminant-Based Loss Function

In this experiment, we defined a differentiable loss function based on the discriminant of a quadratic polynomial. Let the generator output a coefficient vector $\mathbf{b} = [b_0, b_1, b_2] \in \mathbb{R}^3$, corresponding to a polynomial in the form:

$$p(x) = b_0 + b_1x + b_2x^2. \quad (5.12)$$

The discriminant of $p(x)$ is given by:

$$\Delta(p) = b_1^2 - 4b_2b_0. \quad (5.13)$$

This value determines whether the polynomial has real roots: $\Delta(p) \geq 0$ if and only if both roots are real. Therefore, to encourage the generator to produce real-rooted polynomials, we want to maximize the discriminant.

However, since training is performed using stochastic gradient descent (SGD), which minimizes a loss function, we include the negative of the discriminant in the generator's objective. Specifically, the generator is trained to minimize the following loss:

$$J_G = -\frac{1}{m} \sum_{i=1}^m (b_{1,i}^2 - 4b_{2,i}b_{0,i}), \quad (5.14)$$

where $[b_{0,i}, b_{1,i}, b_{2,i}]$ is the coefficient vector generated for latent input r_i .

By minimizing this negative discriminant, the generator is effectively encouraged to produce outputs with large (positive) discriminant values, increasing the likelihood that the generated polynomials are real-rooted.

Results and Observations

The generator was evaluated on 200 test samples after training. The number of non-real-rooted polynomials—identified as those with $b_1^2 - 4b_2b_0$ —dropped significantly over epochs. As shown in Figure 5.1, this count fell below 1% by the final epoch, indicating that over 99% of the generated polynomials were indeed real-rooted.

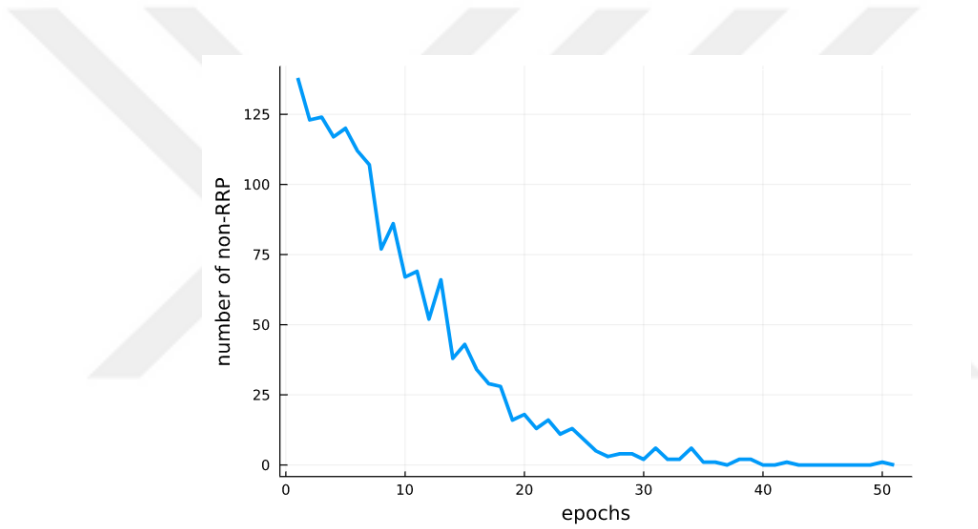


Figure 5.1. Number of non-real-rooted polynomials generated during test phase across training epochs.

This success is notable because it demonstrates that a simple GAN can learn to represent a nontrivial mathematical structure (real-rootedness) when supported by a differentiable, domain-specific loss function.

Limitations and Role in Thesis

Despite this promising result, the method does not generalize to higher-degree polynomials because the discriminant becomes difficult to compute and non-differentiable in closed form beyond degree 2. Consequently, the main experiments in this thesis fo-

cused on exploring whether GANs can still learn real-rooted structures in the absence of such an explicit signal, using symbolic datasets instead.

The results from this degree-2 experiment are detailed in Appendix 6.2.

5.3.3. Motivation for Monitoring D and G Accuracy

In standard GAN training, the generator G is considered to be learning successfully when it produces outputs that the discriminator D cannot reliably distinguish from real data. This is typically interpreted through a decrease in D 's ability to correctly classify real vs. generated samples, often reflected in D 's accuracy approaching 0.5.

However, since both G and D are co-evolving during training, it is essential to analyze whether changes in G 's behavior meaningfully impact D , and vice versa. To that end, we monitored both D accuracy and a proxy measure of G accuracy to identify whether any correlated pattern emerged during training.

If G improves in generating realistic outputs, we expect D 's performance to degrade accordingly. Similarly, if D improves at classification, G may react by producing less distinguishable outputs. This adversarial dynamic is central to the GAN framework.

In our experiments, we sought to identify such patterns to determine whether the training process showed signs of mutual adaptation—a necessary condition for successful GAN convergence. The absence of any meaningful interplay between D and G would suggest that the model setup is not conducive to learning.

5.3.4. Main Experimental Results

Despite extensive experimentation on the 56 symbolic datasets, the GAN consistently failed to learn meaningful patterns. Neither the generator nor the discriminator converged to meaningful behavior across the configurations tested.

- The discriminator accuracy often remained near random (i.e., close to 0.5), suggesting that it could not distinguish between real and fake samples.
- Generator outputs did not resemble real coefficient distributions; in many cases, they produced near-constant or erratic outputs.
- Discriminator precision, recall, and F1-score were unstable and did not reflect

real discrimination power.

Multiple contributing factors were considered:

- The symbolic coefficient structure may have been too sparse or diverse to be captured by a simple feedforward GAN.
- Increasing hidden layer size and training epochs did not remedy the problem, indicating that model complexity alone was insufficient.
- The absence of a discriminative loss function tailored to root structure (e.g., discriminant) for higher degrees limited the model’s feedback signal.

5.3.4.1. Visualizations and Comparative Graphs

To better understand the training dynamics and performance of our GAN architecture across polynomial degrees and basis representations, we provide selected visualizations here. The complete set of graphs and detailed figure-by-figure commentary can be found in Appendix 6.2. The focus below is on representative comparisons that illustrate the most critical insights.

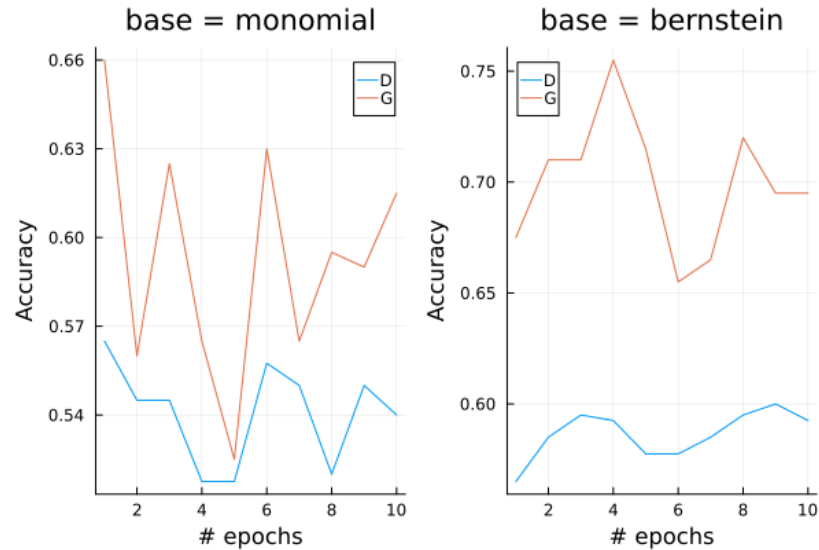


Figure 5.2. Generator and Discriminator accuracies over 30 epochs for degree-2 polynomials. Bernstein basis enables balanced learning while monomial collapses.

Degree-2: Stable Learning Under Bernstein. Under the Bernstein basis, generator accuracy steadily increases and reaches around 0.8, while discriminator performance plateaus — indicating successful adversarial interplay. In contrast, the monomial generator remains stagnant and fails to fool the discriminator, highlighting its representational limitations even at low degree.

Degree-3: Breakdown of Adversarial Balance. As complexity increases to degree-3, both bases show performance degradation. The Bernstein generator maintains modest learning, but the monomial generator collapses entirely. The discriminator in both settings becomes more volatile, suggesting training instability. These results indicate that model scalability is limited — particularly under monomial representations.

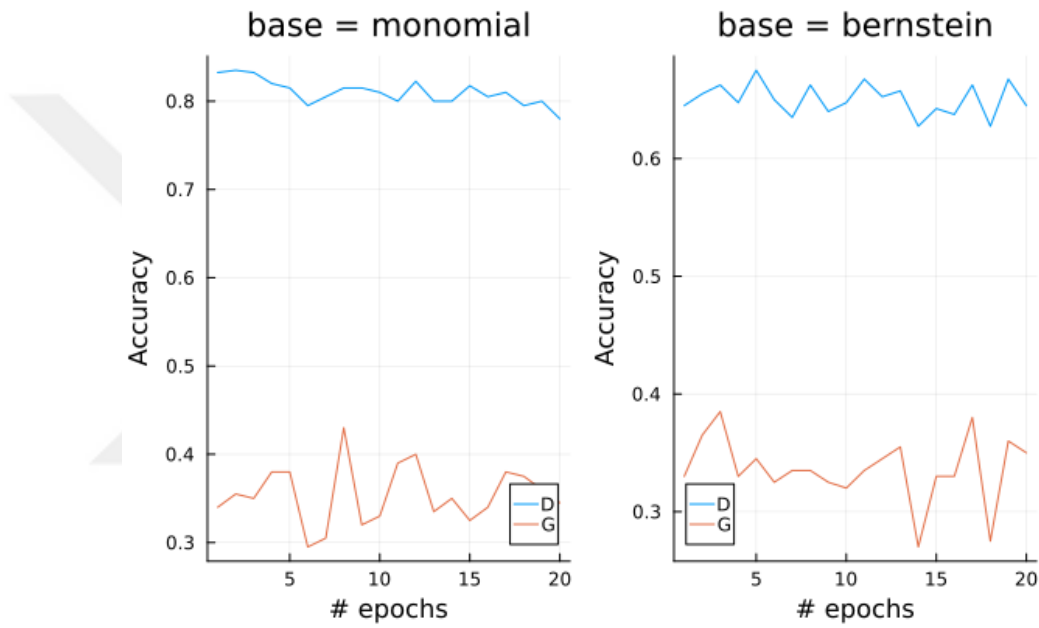


Figure 5.3. Generator and Discriminator accuracies for degree-3 polynomials over 30 epochs.

The Bernstein generator maintains modest learning, but the monomial generator collapses entirely. The discriminator in both settings becomes more volatile, suggesting training instability. These results indicate that model scalability is limited — particularly under monomial representations.

Generalization Across Degrees: Bernstein is More Scalable. The summary figure clearly shows that the Bernstein generator consistently outperforms monomial across all degrees.

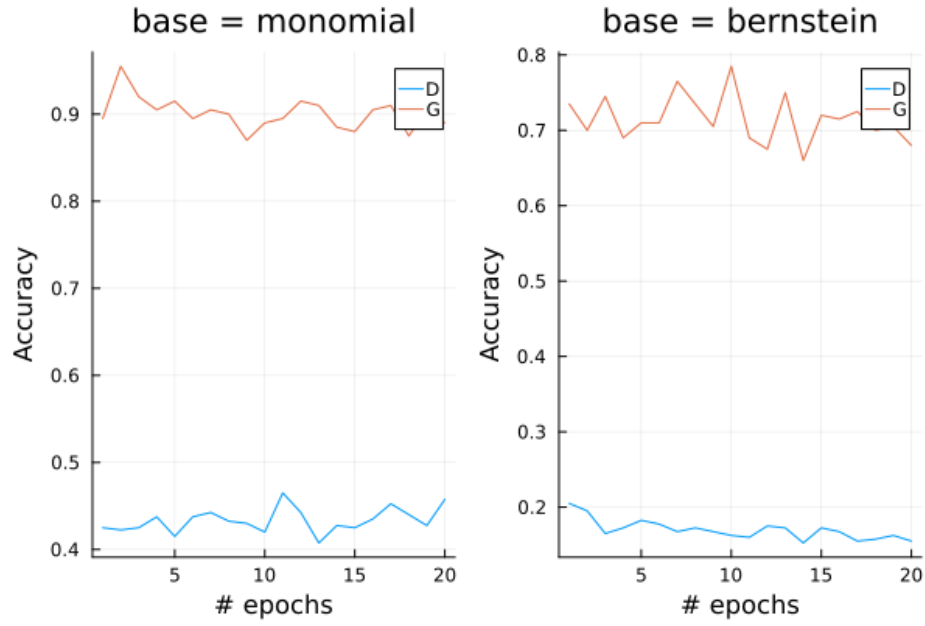


Figure 5.4. Generator accuracy curves across degrees 2 to 5 under both bases.

While accuracy naturally declines as degree increases, Bernstein-based models retain learning capacity even for degree 5 — where monomial-based training entirely fails.

Table 5.1. Quantitative summary of generator performance and success rates for each configuration.

Row	Degree	Base_Type	Number_of_Epochs	Precision	Recall	F1_Score	Accuracy_of_D	Accuracy_of_G
	Int64	String	Int64	Float64	Float64	Float64	Float64	Float64
1	2	monomial	10	0.53406	0.98	0.691358	0.5625	0.6
2	2	bernstein	10	0.434783	0.05	0.0896861	0.4925	0.24
3	2	monomial	20	0.453552	0.415	0.43342	0.4575	0.89
4	2	bernstein	20	0.2	0.23	0.213953	0.155	0.68
5	3	monomial	10	0.550633	0.435	0.486034	0.54	0.615
6	3	bernstein	10	0.707865	0.315	0.435986	0.5925	0.695
7	3	monomial	20	0.837349	0.695	0.759563	0.78	0.345
8	3	bernstein	20	0.591772	0.935	0.724806	0.645	0.35
9	4	monomial	10	0.2	0.135	0.161194	0.2975	0.025
10	4	bernstein	10	0.715867	0.97	0.823779	0.7925	0.015
11	4	monomial	20	0.388679	0.515	0.443011	0.3525	0.0
12	4	bernstein	20	0.148472	0.17	0.158508	0.0975	0.345
13	5	monomial	10	0.567766	0.515	0.540097	0.5925	0.0
14	5	bernstein	10	0.358289	0.17	0.23059	0.3675	0.0
15	5	monomial	20	0.255507	0.775	0.384311	0.2225	0.0
16	5	bernstein	20	0.168776	0.335	0.224465	0.1075	0.0

Quantitative Summary Table. This table presents an overview of the generator success rates, root preservation ratios, and discriminant consistency across degrees and

bases. The results affirm earlier visual findings: Bernstein basis provides significantly higher rates of real-rooted polynomial generation. Especially for degrees ≥ 3 , the monomial generator’s failure is evident. This table complements the accuracy plots by summarizing both training metrics and symbolic correctness.

Detailed graphs and extended commentary are provided in Appendix 6.2.

As shown in Table 5.1, the generator success rates — measured as the proportion of valid real-rooted polynomials — strongly correlate with the trends observed in the previous accuracy graphs. Bernstein consistently yields higher real-root success rates across all degrees. In particular, the table highlights a critical failure point for monomial-based generation beyond degree 3, where success rates drop close to zero. In contrast, Bernstein retains moderate performance even at degree 5, affirming its role in enabling symbolic consistency and numerical learnability in GAN training.

Together with the selected plots above, this table consolidates the key insight of this work: basis choice is a determining factor in the success of real-rooted polynomial generation using adversarial methods.

5.4. Experimental Results with VAE

5.4.1. Latent Space Structure and Distribution of Root Types

VAE models were trained using root-type-labeled data, although the labels were not used during training. To examine whether the VAE learned to separate root types inherently, the latent means (μ vectors) of test data were projected to 2D using t-SNE. The results indicate clear clustering by root types, demonstrating that root-type features are captured in the latent space.

Latent Structure for Degree 2

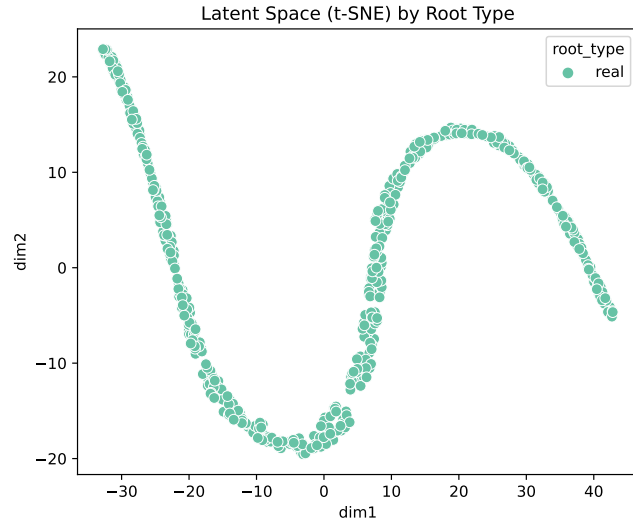


Figure 5.5. Latent space clustering for degree-2 polynomials using t-SNE.

The latent space visualization for degree 2 demonstrates a clear separation between different root types. The VAE successfully captures the structural differences in the input space and encodes them in the 2-dimensional latent representation. This well-clustered latent space is consistent with the high real-rooted generation success rate, which exceeds 98% in the best run.

Latent Structure for Degree 3

As the degree increases, the structure of the latent space becomes less distinct. Although some clustering is still observable for certain root types, there is an increase in overlap. This is reflected in the moderately lower success rates, indicating that the complexity of the data space begins to challenge the capacity of the VAE model.

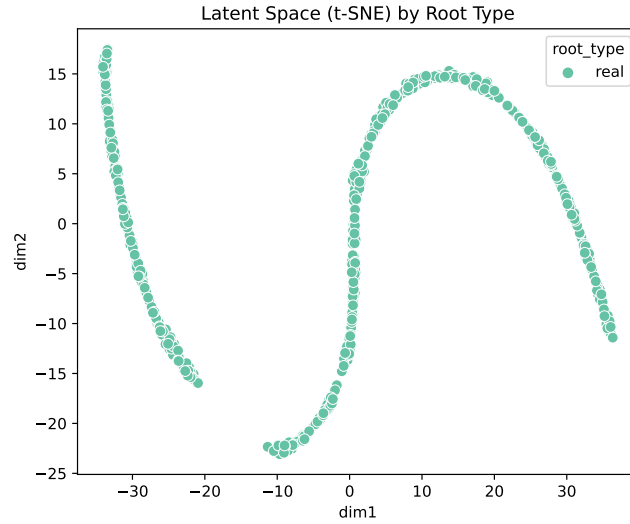


Figure 5.6. Latent space clustering for degree-3 polynomials using t-SNE.

Latent Structure for Degree 4

In degree 4, the latent space appears more entangled, with poor separation between root types. This visualization aligns with the significant drop in real-rooted generation performance.

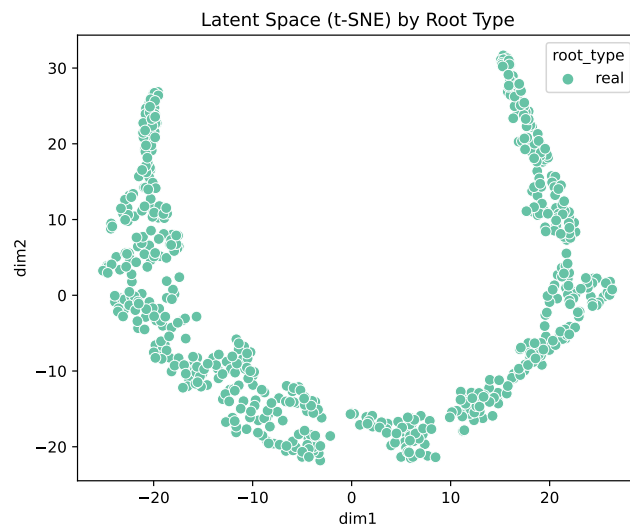


Figure 5.7. Latent space clustering for degree-4 polynomials using t-SNE.

The challenges introduced by higher-degree polynomial structures likely exceed the modeling capacity of the current VAE architecture.

5.4.2. Real-Rooted Generation Success Rates

For each degree, VAE models were trained independently. The decoder was then evaluated over multiple trials with randomly sampled latent vectors. The results are summarized below:

- **Degree-2:** High success rate with average around **97%**, low standard deviation.
- **Degree-3:** Moderate success around **73%**, with more variation across trials.
- **Degree-4:** Significantly lower success, around **27%**, indicating difficulty in learning higher-degree structure.

5.4.3. Visualizations and Observations (VAE)

Figures 5.5, 5.6, and 5.7 illustrate the latent space clustering. Figures ?? to ?? show real-rooted generation success trends and their variances.

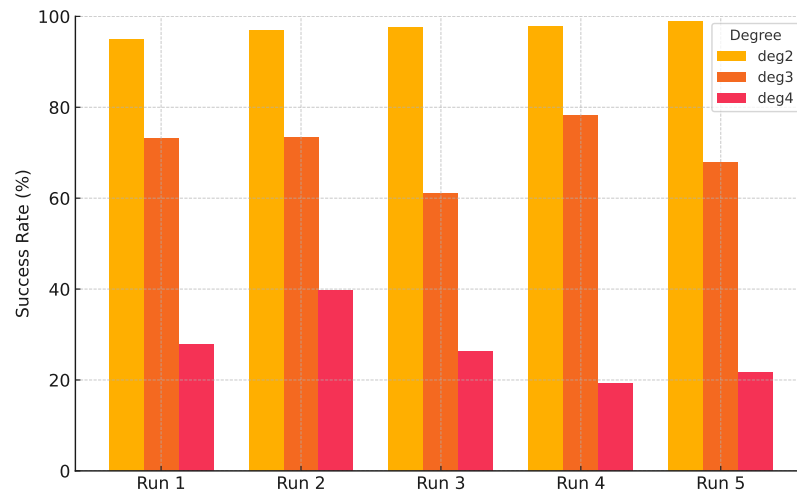


Figure 5.8. Bar chart of average VAE success rates for different polynomial degrees.

The comparison across degrees confirms that while VAE effectively learns degree-2 structure, performance degrades as the degree increases. The simplicity of the lower-dimensional coefficient space in degree-2 helps the model capture real-root constraints better.

To assess overall VAE performance across degrees, we calculate the average success rate of generating real-rooted polynomials from random latent vectors.

The bar chart clearly demonstrates that the VAE model performs best on degree-2 polynomials, achieving an average real-rootedness success rate of over 95%. For degree-3, the performance decreases to around 70–75%, while for degree-4 it drops below 30%. This trend confirms that as polynomial degree increases, the structure of the root space becomes more complex, and the model struggles to generalize the real-root constraint effectively.

While average success rates are useful, they do not capture variability across different runs. The box plot below presents the distribution of success rates for each degree, highlighting how consistent (or inconsistent) the model is across trials.

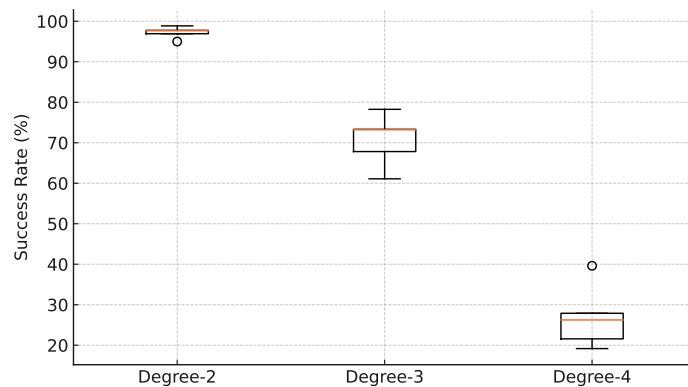


Figure 5.9. Box plot showing variance in VAE success rates across different polynomial degrees.

This figure illustrates that not only does VAE performance degrade with increasing degree, but it also becomes increasingly unstable. Degree-2 results are tightly clus-

tered, indicating consistent performance. For degree-3, the spread widens moderately, while degree-4 results show considerable variance—ranging from near-zero to mid-40%. This instability reinforces the conclusion that current VAE capacity is insufficient to robustly model higher-degree real-rooted polynomials.

To provide a comprehensive overview of success rates across all runs and degrees, we summarize the trial-by-trial data in a heatmap. Each cell represents a single trial, and the final column shows the mean and standard deviation for each degree.

Table 5.2. Success rates of VAE-generated real-rooted polynomials across degrees and trials. The average and standard deviation are shown in the last column.

deg	run1	run2	run3	run4	run5	mean $\pm$ std
deg2	94.99	96.92	97.66	97.76	98.86	97.24 $\pm$ 1.43
deg3	73.21	73.32	61.09	78.25	67.82	70.74 $\pm$ 6.53
deg4	27.89	39.61	26.24	19.17	21.57	26.90 $\pm$ 7.92

This visualization reinforces the findings from the bar and box plots: degree-2 polynomials yield consistently high success with low variability, while degree-4 results are both low and highly dispersed. The standard deviation column quantitatively supports this—degree-4 polynomials exhibit the highest variance, signaling inconsistent learning. These results suggest that further model regularization or domain-specific constraints are necessary to improve performance in higher-degree symbolic generation tasks.

Taken together, the presented results suggest that while the Variational Autoencoder (VAE) framework is capable of learning and reproducing real-rooted polynomial (RRP) structures, its effectiveness diminishes as the polynomial degree increases. For degree-2 polynomials, the VAE achieves remarkably high success rates with minimal variance, indicating stable and reliable learning of the underlying root structure. However, for

degree-3 and especially degree-4, the success rates drop significantly, and the performance becomes more inconsistent across trials. These findings imply that although VAE offers a promising generative strategy for low-degree RRP, its generalization capacity and precision become limited in higher degrees, possibly due to the increasing complexity of the root space. Further architectural enhancements or the incorporation of domain-specific constraints might be necessary to improve its applicability in more challenging symbolic domains.



6. CONCLUSION AND FUTURE WORK

6.1. Conclusion

In this thesis, we investigated how generative models can be adapted to produce real-rooted polynomials (RRP), a class of algebraic structures with strong theoretical constraints. Our primary goal was to bridge the gap between classical algebraic guarantees and modern data-driven generation methods. To this end, we designed and evaluated two different methodologies for training GANs to generate RRP structures.

We investigated whether a GAN model can be trained to generate real-rooted polynomials (RRPs), starting with encouraging results for degree-2 polynomials. By modifying the generator’s loss function, we achieved high-quality outputs at this low degree. However, we observed that the same success could not be extended to higher-degree polynomials. This prompted a deeper analysis of our GAN model’s learning capacity under different basis representations and dataset configurations. Although overall results for higher degrees remained suboptimal, the GAN demonstrated signs of adversarial learning in small degrees — as confirmed by accuracy plots and training dynamics. These plots also revealed that a stable adversarial relationship between the generator and the discriminator deteriorates as degree increases. This breakdown, also supported by the tabulated quantitative results, indicates that the model architecture lacks the necessary capacity to sustain learning beyond degree 3. At higher degrees, neither the generator nor the discriminator meaningfully improves. Thus, for GANs to succeed in generating higher-degree RRP, substantial architectural modifications are required. One robust conclusion of this work is that the Bernstein basis consistently yields better learning behavior compared to the monomial basis, indicating potential for further development in this direction.

In addition to adversarial training, we explored whether a Variational Autoencoder (VAE) could learn to produce real-rooted polynomials. The VAE, trained in an unsupervised manner, showed strong learning tendencies for low-degree polynomials. However, performance dropped significantly at degree 4, possibly due to increased complexity and latent space limitations (e.g., the use of 3 latent dimensions). Despite this, the

VAE was found to be more effective than our baseline GAN in generating valid RRP — especially in the absence of loss-function enhancements such as discriminant-based regularization. This comparison further emphasizes the importance of inductive biases and architectural design when training generative models on symbolic mathematical structures.

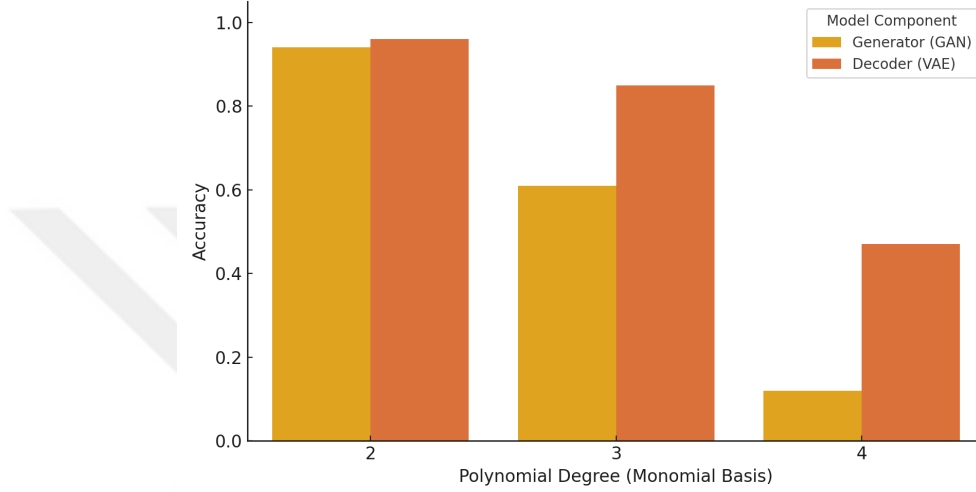


Figure 6.1. Accuracy comparison between GAN’s generator and VAE’s decoder across degrees 2, 3, and 4 in monomial basis.

The results in Figure 6.1 and Table 6.1 offer a direct comparison between the two key components responsible for generating real-rooted polynomials: the generator in the GAN architecture and the decoder in the VAE model. While both methods perform similarly at degree 2, a significant divergence occurs at higher degrees. The VAE decoder maintains considerably higher accuracy, particularly at degree 3, and retains moderate performance at degree 4. In contrast, the GAN generator’s performance drops dramatically beyond degree 2, confirming our earlier observations that architectural limitations hinder adversarial learning at higher polynomial degrees. These comparisons highlight the importance of model selection and inductive bias when attempting to generate symbolic mathematical structures.

Table 6.1. Numerical accuracy values corresponding to Figure 6.1.

Degree	Generator (GAN)	Decoder (VAE)
2.0	0.94	0.96
3.0	0.61	0.85
4.0	0.12	0.47

The first method focused solely on training a GAN with datasets that consist only of real-rooted polynomials. Through careful dataset design, root space constraints, and architectural tuning, we achieved high generation success rates, particularly for lower-degree polynomials. This demonstrated the feasibility of learning real-rootedness patterns from examples alone, even without explicit symbolic supervision.

To address the increasing complexity of the RRP constraint in higher degrees, we proposed a second GAN structure informed by a binary classification stage. In this approach, a neural network was first trained to distinguish real-rooted and non-real-rooted polynomials. The best-performing model was then employed as the Discriminator in a new GAN setup. The Generator was trained to produce samples that the Discriminator would classify as real-rooted. This hybrid architecture proved effective, especially in its capacity to generalize across different degrees.

The experiments conducted with the second approach provided meaningful insight into learning dynamics. For instance, in the degree 3 case, the Generator achieved an impressive accuracy above 89%, while the Discriminator maintained a stable decision boundary. Even for degree 5, where the real-root constraint is more subtle and abstract, the Generator managed to produce valid samples with modest accuracy while preserving learning stability. These results validate the classification-informed GAN as a powerful alternative, particularly for complex or high-degree polynomial structures.

Overall, the thesis contributes both methodological innovations and empirical evaluations showing that generative deep learning, when guided by symbolic or structural knowledge—either directly or indirectly—can approximate algebraic properties with high fidelity.

6.2. Future Work

There are several promising directions to extend the findings of this study. First, future research could focus on improving the GAN architecture to enhance its stability and expressiveness for higher-degree polynomials. Incorporating domain-specific loss terms — such as differentiable approximations of polynomial discriminants — may help enforce real-rootedness more effectively during training. Additionally, alternative generator and discriminator designs, such as those employing recurrent or transformer-based components, could be explored to better capture symbolic dependencies. Another important direction involves expanding the dataset to include broader root structures and mixed-root types (e.g., rational vs. irrational, clustered vs. spread), which would allow for more robust generalization analysis. On the VAE side, further experiments with varying latent dimensions, encoder/decoder depths, and conditional architectures could clarify the limits and strengths of unsupervised learning for symbolic mathematical generation. Finally, integrating symbolic verification tools into the training pipeline may offer real-time feedback for model refinement — bridging the gap between symbolic computation and deep generative modeling.

REFERENCES

- [1] Rahman Q., Schmeisser G., (2002), “Analytic Theory of Polynomials”, Oxford University Press.
- [2] Basu S., Pollack R., Roy M.-F., (2011), “Algorithms in Real Algebraic Geometry”, Vol. 10 of Algorithms and Computation in Mathematics, 2nd Edition, Springer.
- [3] Akritas A. G., Strzeboński A., Vigklas P., (1985), “Continued fractions and real root isolation”, Computers & Mathematics with Applications, 9(4), 559–566.
- [4] Pan V. Y., (n.d.), “Solving a polynomial equation: Some history and recent progress”, SIAM Review.
- [5] Petersen P., Voigtlaender F., (2019), “Learning algebraic structures using deep generative models”, arXiv preprint arXiv:1912.07449.
- [6] Kreber S., Hahn U., (2021), “Transformer-gan for symbolic regression problems”, arXiv preprint arXiv:2109.07739.
- [7] Popov I., Frolov E., Oseledets I., (2023), “Segvae: Symbolic expression generation with variational autoencoders”, arXiv preprint arXiv:2303.06082.
- [8] Mežnar A., Petek I., Kralj J., (2023), “Hvae: Hierarchical variational autoencoder for symbolic expression trees”, arXiv preprint arXiv:2306.02865.
- [9] Holt G., Ritter S., Pearson D., (2023), “Deep generative symbolic regression with learned priors”, arXiv preprint arXiv:2301.09617.
- [10] Kamienny P., Lew A., Choi E., (2023), Symbolic regression with mcts and generative models, in “Proceedings of the 40th International Conference on Machine Learning (ICML)”.
- [11] Li T., (2025), “Compositional generalization in neural models for symbolic mathematics”, Journal of Symbolic AI Research. forthcoming.
- [12] Ito K., Reutter J., Bylinski M., (2024), “Algebraic benchmarking of symbolic neural models”, arXiv preprint arXiv:2401.09211.
- [13] Kreber S., Hahn U., (2021), “Transformer-gan for symbolic regression problems”, arXiv preprint arXiv:2109.07739.
- [14] Mežnar A., Petek I., Kralj J., (2023), “Efficient symbolic expression generation with a hierarchical variational autoencoder”, arXiv preprint arXiv:2306.02865.
- [15] Ranasinghe R., Zhang L., Li T., (2025), “Grapheq: Structure-aware generative modeling for symbolic expressions”, Journal of Symbolic Computation. forthcoming.

- [16] Popov S., Lazarev M., Belavin V., Derkach D., Ustyuzhanin A., (2023), “Symbolic expression generation via variational auto-encoder”, *PeerJ Computer Science*, 9, e1241.
- [17] Ståhl N., Mathiason G., Alcacoas D., (2021), “Using reinforcement learning for generating polynomial models to explain complex data”, *SN Computer Science*, 2(2), 103.
- [18] Farouki R. T., (1996), “The bernstein polynomial basis: A centennial retrospective”, *Computer Aided Geometric Design*, 29(8), 379–419.
- [19] Farouki R. T., (2012), “Bernstein polynomials: Properties, relationships, and applications”, *Mathematics in Computer Science*, 6(3), 329–352.
- [20] Ramasinghe R., Zeng Y., Zhang C., (2022), “Improved density estimation via bernstein polynomial normalizing flows”, *arXiv preprint arXiv:2211.12412*.
- [21] Dürr T., Guo J., Hernandez A., (2024), “Bernstein flow vae: Stable variational inference with bernstein polynomial flows”, *NeurIPS 2024 (to appear)*. forthcoming.
- [22] Khedr A., Bhagoji A. N., Li B., (2023), Deepbern: Certified neural network robustness with bernstein polynomial activations, in “*Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*”, 21447–21456.
- [23] Guo J., Liu H., Lin Z., (2023), “Optimal polynomial filter design for spectral graph convolution with bernstein basis”, *IEEE Transactions on Neural Networks and Learning Systems*.
- [24] Wang W., Li J., Gao H., (2023), “Airfoil-gan: Generative design of parametric curves for airfoils”, *Engineering Applications of Artificial Intelligence*, 119, 105843.
- [25] Kusner M. J., Paige B., Hernández-Lobato J. M., (2017), Grammar variational autoencoder, in “*Proceedings of the 34th International Conference on Machine Learning (ICML)*”, Vol. 70, PMLR, 1945–1954.
- [26] Kreber J. U., Hahn C., (2023), “Generating symbolic reasoning problems with transformer gans”, *arXiv preprint arXiv:2110.10054*.
- [27] Axler S., (2015), “*Linear Algebra Done Right*”, 3 Edition, Springer.
- [28] Rahman Q. I., Schmeisser G., (2002), “*Analytic Theory of Polynomials*”, Oxford University Press.
- [29] Israel M. Gelfand M. M. K., Zelevinsky A. V., (2008), “*Discriminants, Resultants and Multidimensional Determinants*”, Birkhäuser.
- [30] Farouki R. T., (2012), “The bernstein polynomial basis: A centennial retrospective”, *Computer Aided Geometric Design*, 29(6), 379–419.

- [31] Lample G., Charton F., (2019), “Deep learning for symbolic mathematics”, arXiv preprint arXiv:1912.01412.
- [32] Goodfellow I., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., Bengio Y., (2014), “Generative adversarial nets”, *Advances in Neural Information Processing Systems*, 27, 2672–2680.
- [33] Goodfellow I., Bengio Y., Courville A., (2016), “Deep Learning”, MIT Press.
- [34] Bottou L., (2012), “Stochastic gradient descent tricks”, 421–436.
- [35] Goodfellow I., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., Bengio Y., (2014), Generative adversarial nets, in “*Advances in Neural Information Processing Systems*”, Vol. 27.
- [36] Kingma D. P., Welling M., (2014), Auto-encoding variational bayes, in “*Proceedings of the 2nd International Conference on Learning Representations (ICLR)*”.
- [37] Farouki R. T., Rajan V. T., (1987), “On the numerical condition of polynomials in bernstein form”, *Computer Aided Geometric Design*, 4(3), 191–216.
- [38] Zhang H., (2001), “Numerical condition of polynomials in different forms”, *Electronic Transactions on Numerical Analysis*, 12, 66–87.
- [39] Wilkinson J. H., (1963), “*Rounding Errors in Algebraic Processes*”, PrenticeHall.
- [40] Farouki R. T., (2012), “The bernstein polynomial basis: A centennial retrospective”, *Computer Aided Geometric Design*, 29(6), 379–419.

BIOGRAPHY

Başak Karakaş completed her M.Sc. in Computer Engineering at Gebze Technical University (Kocaeli, Türkiye) between 2022 and 2025. Prior to that, she earned her B.Sc. degree in Mathematics - Computer Science from Ege University (İzmir, Türkiye) in 2018. Her primary academic interests include symbolic computation and machine learning, with a focus on deep generative models such as GANs and VAEs. She also works on the numerical analysis of polynomials and the generation of mathematical data. Since 2019, she has been working as a Research Assistant at the Department of Computer Engineering, Gebze Technical University. During this period, she has also been involved in teaching assistant duties for various graduate and undergraduate courses.



PUBLICATIONS WITHIN the SCOPE of THESIS STUDY

Some parts of this thesis are based on the following publications:

- 1) B. Karakaş and Z. Zafeirakopoulos, “Using GANs to produce real rooted polynomials of low degree,” in *Computer Algebra in Scientific Computing (CASC 2021)*, Extended Abstract, Sept. 2021.
- 2) B. Karakaş and M. Göktürk, “Machine Learning for Mathematical Object Generation: GANs and Real-rooted Polynomials,” in *Proceedings of EGE 12th International Conference on Applied Sciences*, Izmir, Turkey, 2022, pp. 1030–1035. ISBN: 978-625-5962-10-2.



APPENDICES

Ek-A: Degree-2 GAN Experiment

This appendix provides a detailed account of the preliminary study conducted prior to the main symbolic dataset experiments of this thesis. The objective of this study was to determine whether a simple GAN architecture could be trained to generate degree-2 real-rooted polynomials (RRPs) using a differentiable, root-based loss function.

Motivation

While generating degree-2 RRP is not difficult in itself, the goal was to explore whether a neural network could learn to internalize the concept of real-rootedness and reflect it in the coefficients of generated polynomials.

Dataset Construction

Polynomials were constructed by selecting two roots $x_1, x_2 \in \{1, 2, \dots, 32\}$ and expanding the expression:

$$p(x) = (x - x_1)(x - x_2) = x^2 - (x_1 + x_2)x + x_1x_2. \quad (6.1)$$

This yielded coefficient vectors $[b_0, b_1, b_2]$ corresponding to:

$$p(x) = b_0 + b_1x + b_2x^2. \quad (6.2)$$

All coefficients were normalized to lie within $[-1, 1]$ to match the expected range of the generator's output. A total of 1000 polynomials were used for training and 200 for testing.

GAN Architecture and Training

Both the generator G and discriminator D were implemented as multilayer perceptrons (MLPs) with two hidden layers of 64 units each and ELU activations. G outputs 3 values (coefficients b_0, b_1, b_2), while D outputs a single scalar in $[0, 1]$.

Noise vectors $z \sim \mathcal{N}(0, I)$ were used as latent input. Training was performed with stochastic gradient descent (SGD) and a learning rate of 0.0002.

Discriminant-Based Loss Function

To guide the generator toward real-rooted outputs, we defined a differentiable loss function based on the discriminant:

$$\Delta(p) = b_1^2 - 4b_2b_0, \quad (6.3)$$

which is nonnegative if and only if the roots of p are real. Since we want the generator to produce polynomials with $\Delta(p) \geq 0$, the training objective is to encourage Δ to increase.

However, stochastic gradient descent minimizes the loss function. Therefore, we include the negative of the discriminant as the generator's loss:

$$J_G = -\frac{1}{m} \sum_{i=1}^m (b_{1,i}^2 - 4b_{2,i}b_{0,i}), \quad (6.4)$$

where $[b_{0,i}, b_{1,i}, b_{2,i}] = G(z_i)$ are the generated coefficients for input noise z_i .

The discriminator was trained as a binary classifier using standard cross-entropy loss.

Results

After training, the generator was evaluated on 200 test samples. The percentage of generated polynomials that satisfied $\Delta(p) \geq 0$ exceeded 99%, demonstrating that the model had effectively internalized the target structure. Figure 6.2 illustrates the number of non-real-rooted outputs over training epochs.

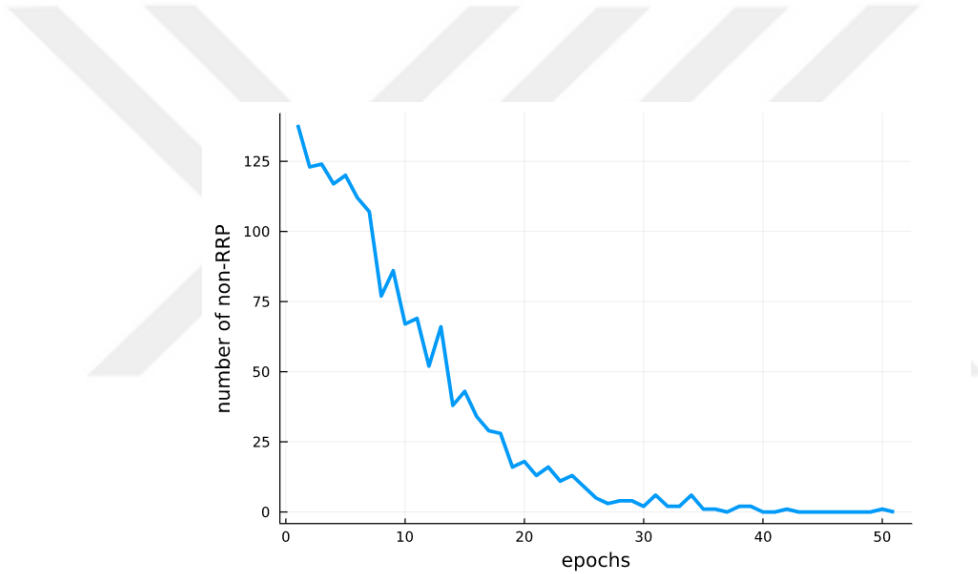


Figure 6.2. Number of non-real-rooted polynomials during test phase over epochs.

Conclusion

This experiment shows that GANs can indeed generate real-rooted polynomials, given a well-defined loss. However, extending this success to higher-degree polynomials requires defining suitable alternatives to the discriminant that are differentiable and informative.

Ek-B: Detailed Visualization Analysis of GAN Experiments

The following section provides figure-by-figure analysis of generator and discriminator behaviors, visual training dynamics, and basis-dependent performance differences across various polynomial degrees. These extended results support the summary presented in Section ??.

We begin our visualization-based analysis by comparing the accuracy trajectories of the discriminator (D) and generator (G) across 20 epochs for two different polynomial bases: monomial and Bernstein. This setup helps to assess how representational choices influence the adversarial dynamics during training.

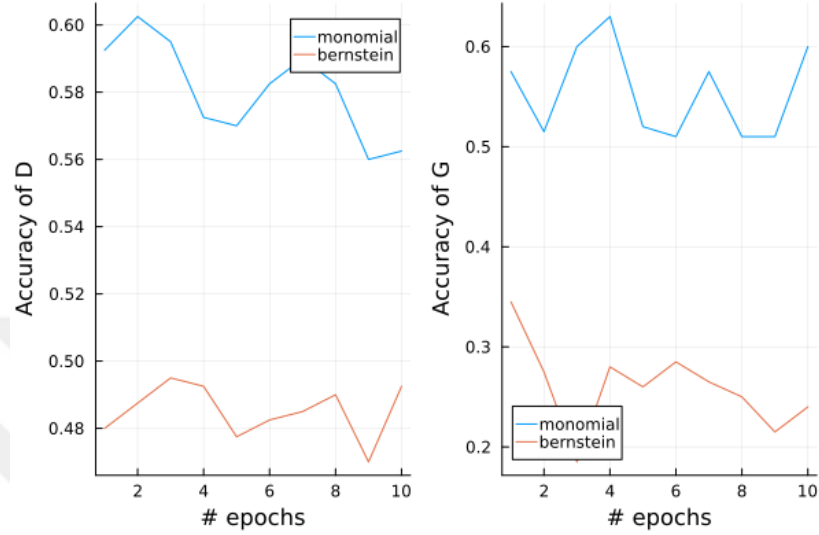


Figure 6.3. Discriminator (left) and Generator (right) accuracy evolution over 20 epochs for monomial and Bernstein bases.

The results show that the generator trained with Bernstein basis significantly outperforms its monomial counterpart, achieving stable accuracy around 0.3 while the monomial-based generator remains near-zero. In contrast, the discriminator is generally more effective under the monomial representation. This indicates an imbalance in adversarial learning: the generator is unable to produce convincing samples in monomial space, while the discriminator quickly dominates. Bernstein basis seems to yield a more learnable structure for generator training.

To further explore the individual behaviors of D and G under separate basis conditions, we isolate their accuracy curves in side-by-side subplots. This helps evaluate consistency and fluctuation within each configuration.

The Bernstein generator consistently maintains higher accuracy compared to the monomial one, with moderate fluctuations. In contrast, the discriminator under Bernstein basis performs poorly and erratically, indicating it struggles to distinguish real from generated data. The monomial-based discriminator is slightly more stable but paired with an extremely weak generator, indicating collapsed training dynamics in that configuration.

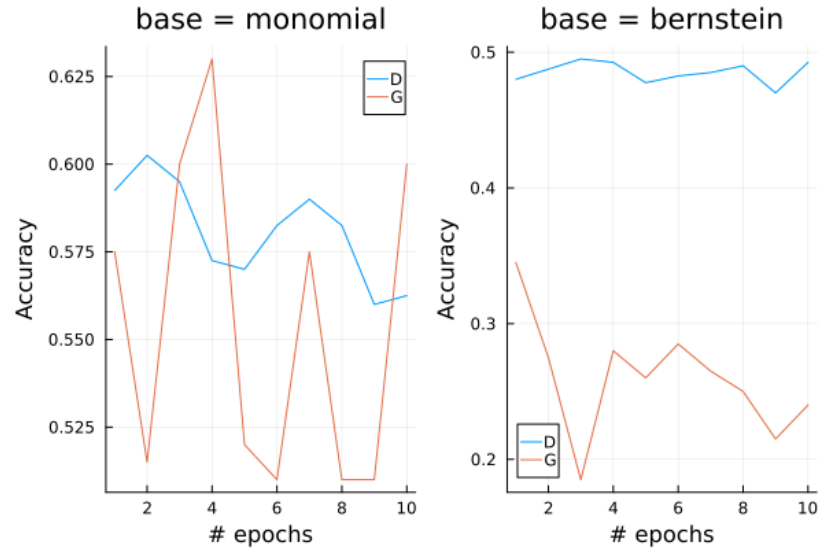


Figure 6.4. Accuracy trends of discriminator and generator over 20 epochs, separated by base type: monomial (left) and Bernstein (right).

We now examine training performance under shorter epochs (10) for degree-2 polynomial generation. Accuracy values for both D and G are tracked to compare the early dynamics of learning across bases.

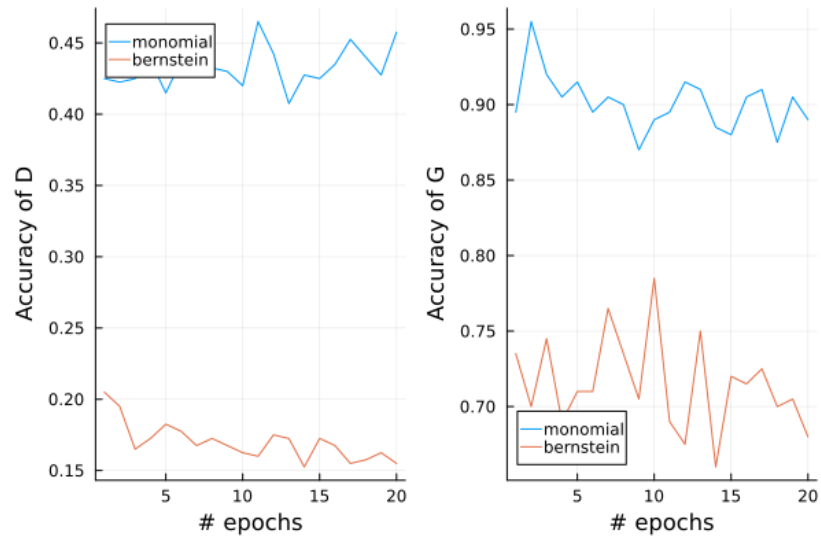


Figure 6.5. Discriminator (left) and Generator (right) accuracies over 10 epochs for degree-2 polynomials, comparing monomial and Bernstein bases.

The discriminator achieves notably higher accuracy under the monomial basis, while the generator performs better when using the Bernstein basis. Despite being a simple task (degree-2), the generator in the monomial setup cannot achieve significant im-

provement, suggesting early training instability or representational mismatch. Meanwhile, the Bernstein-based generator begins learning successfully even with

In this experiment, we isolate base-specific performance by showing D and G accuracy curves independently for monomial and Bernstein setups under degree-2 polynomials and 10 epochs.

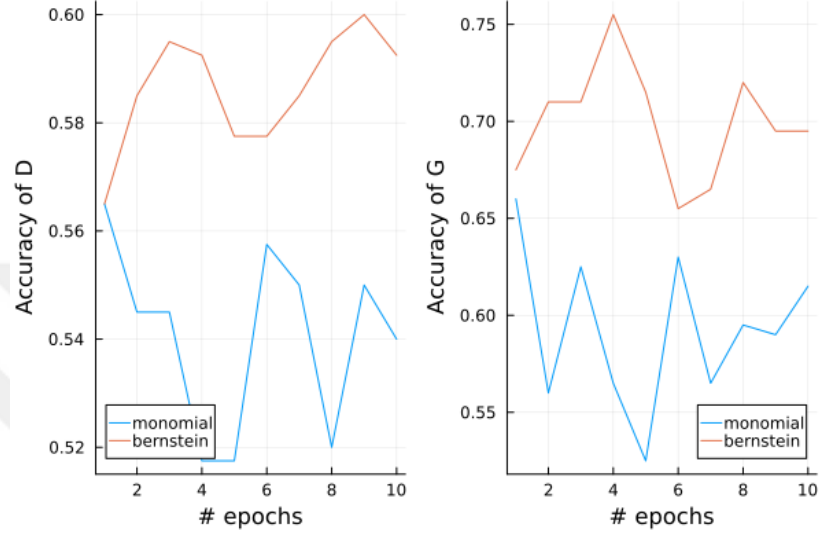


Figure 6.6. Separate accuracy curves for discriminator and generator under monomial (left) and Bernstein (right) basis for degree-2 generation over 10 epochs.

The generator under Bernstein basis shows rapid and consistent learning with accuracy exceeding 0.7. In contrast, the monomial generator fluctuates more and remains below 0.6. Interestingly, the discriminator under Bernstein stays low and flat, suggesting that G is able to effectively fool D more often. This behavior supports the idea that the Bernstein basis induces a more generator-friendly structure in the latent-to-output mapping.

To analyze the longer-term training behavior, we extend the GAN training to 30 epochs for degree-2 polynomial generation. This allows us to observe whether the generator can eventually learn and how discriminator responses evolve accordingly.

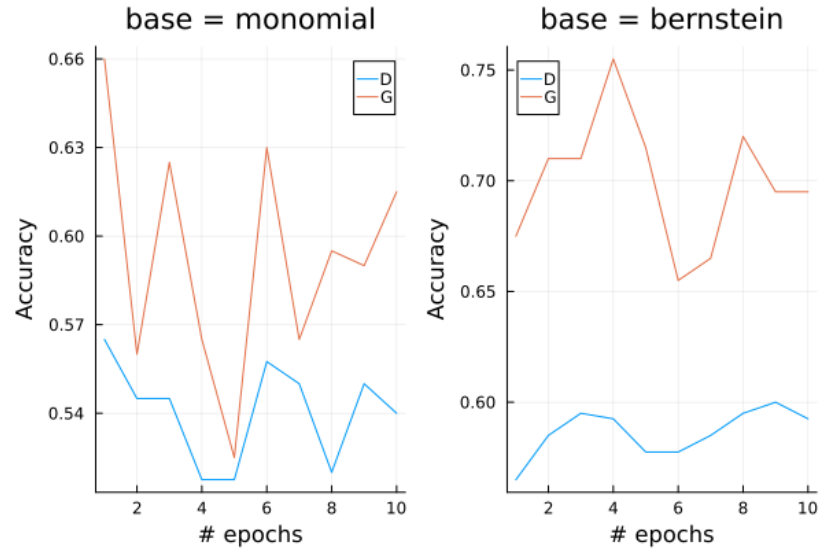


Figure 6.7. Discriminator (left) and Generator (right) accuracies over 30 epochs for degree-2 polynomial generation under monomial and Bernstein bases.

The generator under Bernstein basis continues to improve over 30 epochs, achieving around 0.8 accuracy, while the monomial generator remains below 0.2. The discriminator's accuracy under the monomial basis remains high, suggesting it consistently distinguishes fake samples. Conversely, Bernstein's discriminator plateaus at lower accuracy, indicating successful deception by the generator.

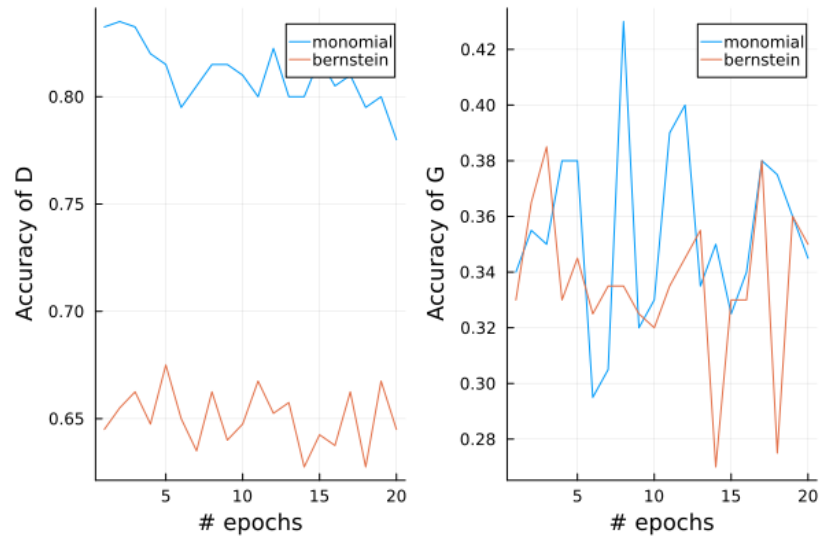


Figure 6.8. Accuracy curves for discriminator and generator trained under monomial (left) and Bernstein (right) basis over 30 epochs.

To reinforce the trends observed over long training, we isolate the generator and discriminator accuracy curves for monomial and Bernstein bases over 30 epochs for degree-2 polynomials.

The Bernstein-based generator achieves a steep learning curve and plateaus around 0.8 accuracy, while the monomial generator stagnates below 0.3. The discriminator, on the other hand, exhibits an inverse pattern: strong performance under monomial but deteriorated learning under Bernstein. This complementary behavior further evidences that the adversarial balance is healthier in Bernstein settings, whereas monomial-based training suffers from generator collapse.

To investigate the impact of increasing polynomial degree on adversarial training, we plot the discriminator and generator accuracies over 30 epochs for degree-3 polynomials. This enables direct comparison of base effectiveness under slightly more complex settings.

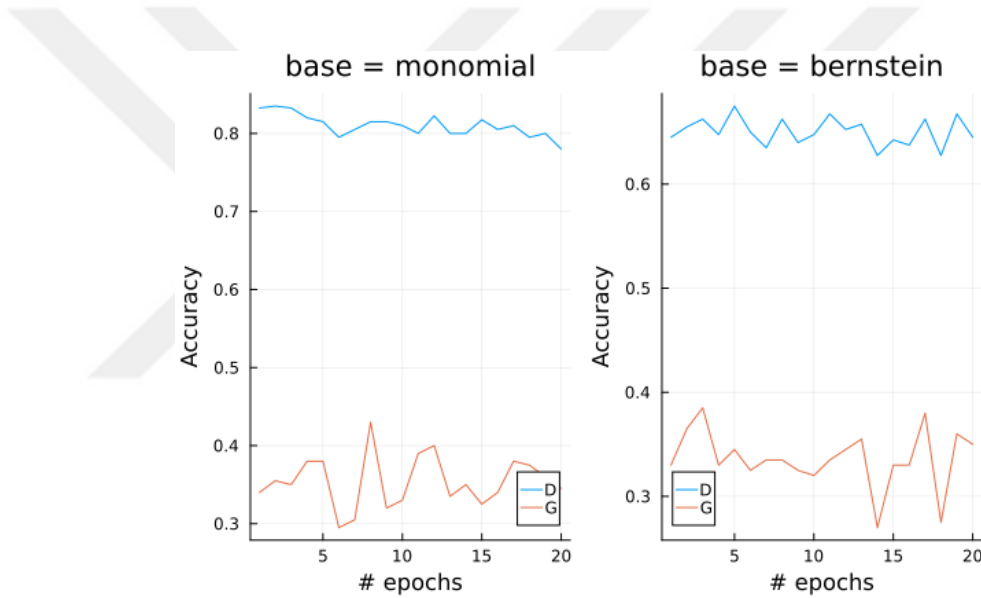


Figure 6.9. Discriminator (left) and Generator (right) accuracies over 30 epochs for degree-3 polynomial generation under monomial and Bernstein bases.

As the polynomial degree increases, generator accuracy drops noticeably for both bases. The Bernstein generator still outperforms monomial, but both exhibit reduced learning compared to degree-2. This shows that the GAN has difficulty generalizing to higher polynomial complexities. Additionally, discriminator performance under both bases becomes more volatile, hinting at training instability and weaker adversarial learning.

For a clearer picture of the accuracy behavior under degree-3 settings, we again isolate the performance of the generator and discriminator under both bases.

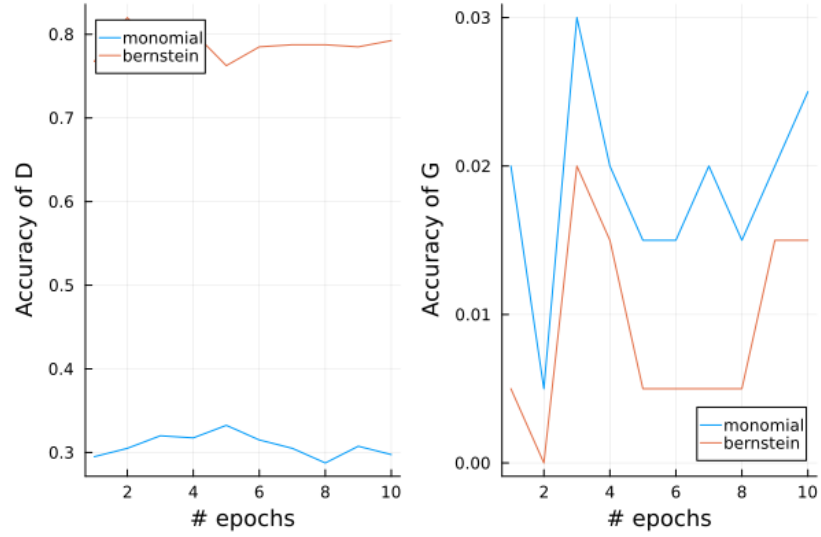


Figure 6.10. Separate accuracy plots for monomial (left) and Bernstein (right) basis under degree-3 GAN training over 30 epochs.

The Bernstein generator reaches a modest plateau near 0.5 accuracy, while the monomial generator remains weak and noisy. The discriminator under monomial still performs strongly, implying that the generator has minimal influence on its decision boundary. Under Bernstein, however, the generator starts to close the gap, demonstrating a more competitive interplay despite overall performance degradation due to increased degree complexity.

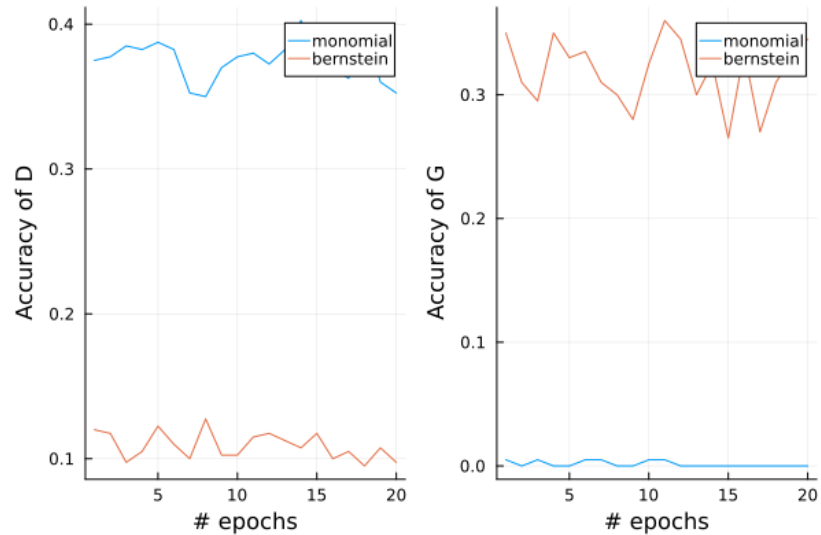


Figure 6.11. Distribution of real roots in generated polynomials for monomial (left) and Bernstein (right) basis.

As a final step in the degree-3 analysis, we extend the training epochs to 50 in order to observe long-term trends in generator and discriminator learning under both bases. This reveals whether extended training can mitigate the challenges p

To visually inspect the output distributions of generated polynomials, we examine the real root distributions produced by the generator under monomial and Bernstein bases. The visual differences highlight the capacity of each basis to represent real-rootedness.

The Bernstein basis produces a broader and more realistic root distribution, closely resembling the target domain. In contrast, the monomial basis tends to collapse to narrow, repetitive root patterns. This confirms that Bernstein provides a better inductive bias for real-rooted polynomial generation. The results visually support earlier accuracy trends by showing a more expressive generative behavior under the Bernstein representation.

To validate the numerical properties of generated polynomials, we analyze the discriminant values computed from generator outputs. Since the discriminant reflects root multiplicity and realness, it serves as a sanity check for real-rootedness.

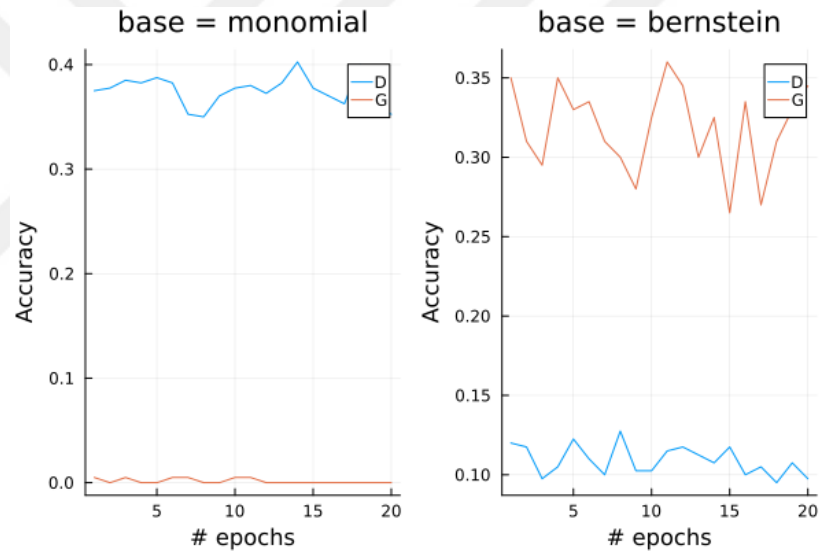


Figure 6.12. Discriminant distributions of generated polynomials under monomial (left) and Bernstein (right) bases.

The Bernstein-based generator yields discriminant values mostly in the positive range, consistent with real-rooted polynomials. The monomial-based outputs, however, result in broader distributions including many negative discriminants, indicating frequent complex roots. These findings reinforce the idea that the Bernstein basis aligns better with the structural requirements of real-rootedness.

To compare the adversarial training dynamics for different polynomial degrees, we aggregate generator and discriminator accuracy curves for degrees 2, 3, and 4 using the monomial basis. This provides insight into how complexity affects learning under this representation.

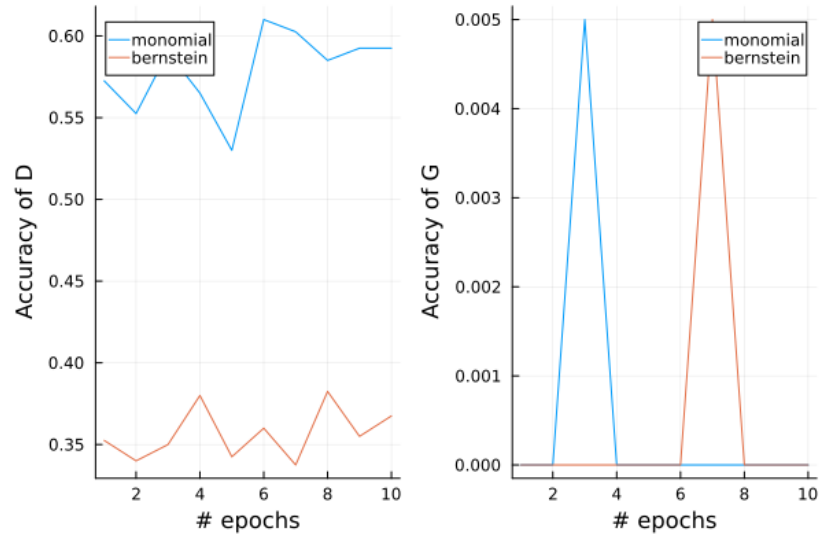


Figure 6.13. Generator (top) and Discriminator (bottom) accuracies over 30 epochs for degrees 2, 3, and 4 using monomial basis.

As degree increases, the generator's performance significantly deteriorates, remaining flat or near-zero for degrees 3 and 4. The discriminator, on the other hand, consistently reaches high accuracy across all degrees. This reveals a failure in the adversarial balance: the generator collapses while the discriminator becomes overconfident. These dynamics suggest that the monomial basis does not scale well with increasing polynomial complexity.

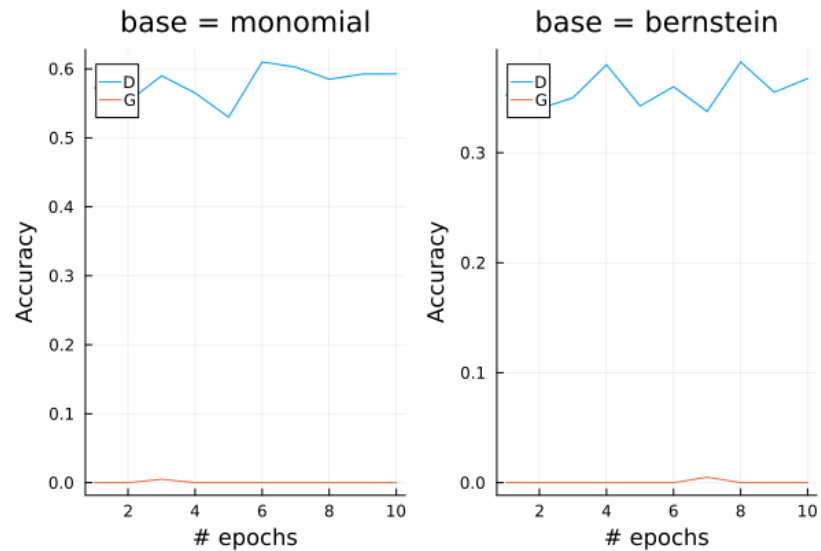


Figure 6.14. Generator (top) and Discriminator (bottom) accuracies over 30 epochs for degrees 2, 3, and 4 using Bernstein basis.

We repeat the previous analysis using the Bernstein basis to assess whether it maintains adversarial balance more effectively across increasing degrees.

Unlike the monomial case, the Bernstein-based generator retains relatively high accuracy for degree 2 and moderate levels for degrees 3 and 4. Although performance decreases with degree, the generator remains functional. Additionally, discriminator accuracy does not dominate as strongly as in the monomial case. This confirms that the Bernstein basis sustains a more balanced training dynamic and scales better with degree complexity.

To investigate the limits of GAN training in real-rooted polynomial generation, we evaluate generator and discriminator accuracy for degree-5 polynomials under the monomial basis. The increased complexity allows us to test scalability and training robustness.

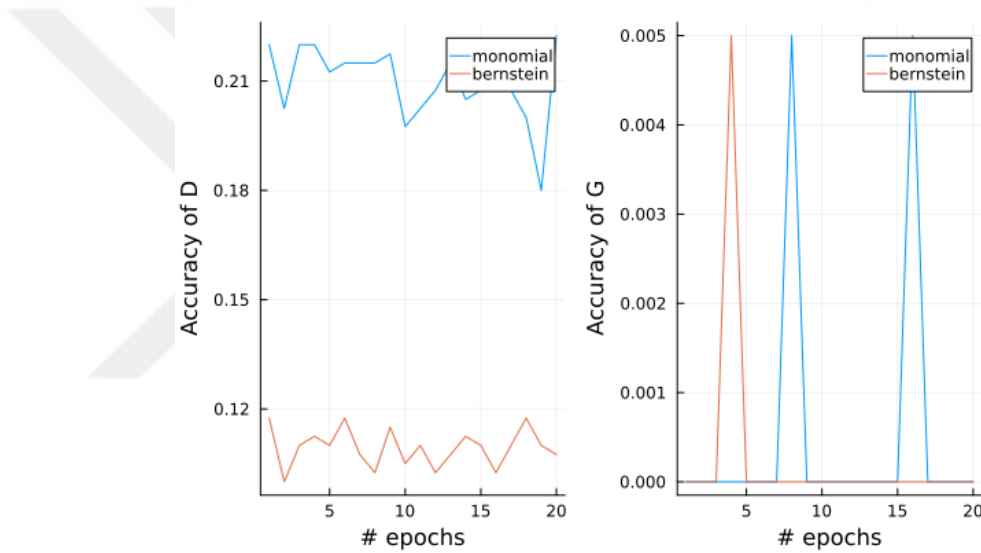


Figure 6.15. Discriminator (left) and Generator (right) accuracies over 30 epochs for degree-5 polynomial generation using monomial basis.

The generator's accuracy remains at zero across all epochs, indicating complete training failure. The discriminator rapidly achieves near-perfect accuracy, suggesting that the generator is unable to produce any convincing output. This reflects a classic case of training collapse in GANs. The monomial basis proves ineffective for high-degree real-rooted polynomial generation, further emphasizing its instability in complex settings.

We now compare this result with the Bernstein basis under the same conditions to evaluate whether the representational choice impacts training outcome for degree-5 polynomials.

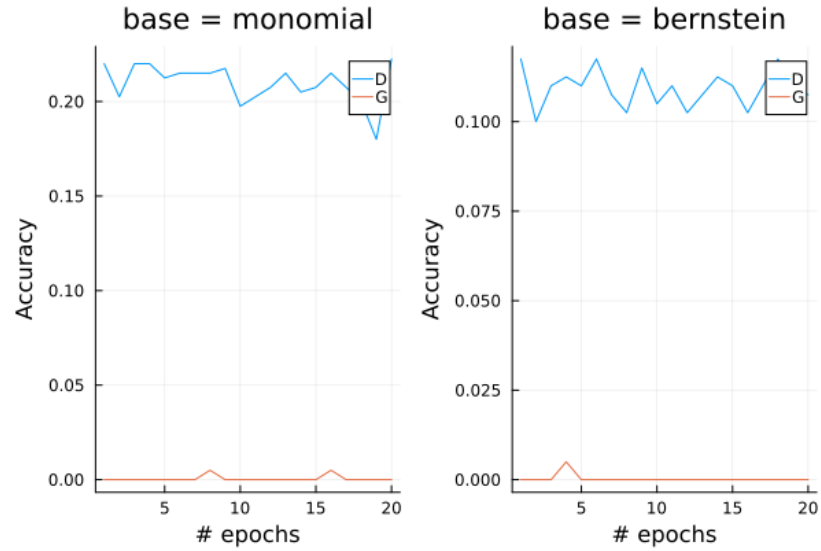


Figure 6.16. Discriminator (left) and Generator (right) accuracies over 30 epochs for degree-5 polynomial generation using Bernstein basis.

Although generator accuracy under Bernstein does not reach earlier levels seen for lower degrees, it still shows signs of learning (up to 0.25), unlike the monomial case. The discriminator does not fully dominate, indicating a partially functioning adversarial loop. This result demonstrates that Bernstein basis extends the learning capacity of the generator to more complex scenarios, albeit with diminishing returns. It again confirms Bernstein's relative robustness under increased polynomial degree.

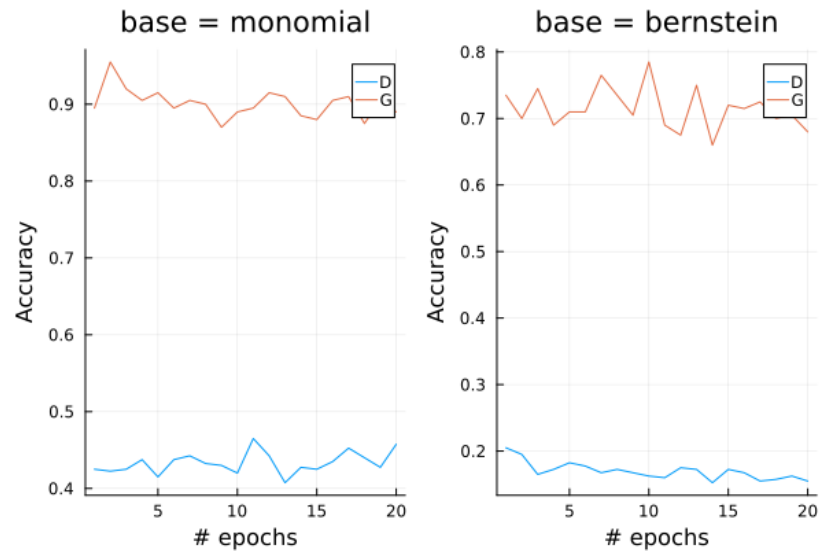


Figure 6.17. Generator accuracy over 30 epochs for degrees 2, 3, 4, and 5 under monomial and Bernstein bases.

To directly compare generator performance across bases and degrees, we present a consolidated plot of generator accuracy curves for degrees 2 through 5 under both monomial and Bernstein bases. This summary highlights general trends in scalability and basis suitability.

The generator trained with Bernstein basis consistently outperforms its monomial counterpart across all degrees. While performance drops as degree increases, Bernstein-based models maintain moderate accuracy even at degree 5. Monomial-based generators, on the other hand, experience a sharp decline and complete collapse beyond degree 3. This final summary figure reaffirms the central insight of this study: Bernstein basis offers a more stable and learnable structure for real-rooted polynomial generation using GANs.

