Pinnacle Academic Press Proceedings Series

Vol. 1 2025

Article Open Access

Enhancing Polynomial Function Education in Secondary Mathematics through Python-Based Computational Tools: A Dual Approach to Theoretical Learning and Practical Application

Yixuan Li [1,*]

[1] Shoreland Lutheran High School, Somers, Kenosha, WI, USA
[*] Correspondence: Yixuan Li, Shoreland Lutheran High School, Somers, Kenosha, WI, USA

PAP Proceedings Series
EEAS·2025
2025 International Conference on Global Economy and Development of Education, Arts, Computer Science and Social Sciences
Chicago·USA | 27-29·March·2025
2025 Vol. 1 ISSN

Received: 17 March 2025
Revised: 22 March 2025
Accepted: 09 April 2025
Published: 12 April 2025

Copyright: © 2025 by the authors. Submitted for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

Abstract: This paper explores the integration of Python-based computational tools in secondary mathematics education, specifically for enhancing the understanding and application of polynomial functions. It addresses the challenges posed by traditional teaching methods, which often involve complex mathematical concepts beyond the typical secondary curriculum. By implementing Python tools that allow students to input equations and receive immediate computational assistance, the study demonstrates how these technologies can support and simplify polynomial function education. This approach not only makes learning more accessible but also aligns with contemporary educational practices that emphasize interactivity and a student-centered approach to learning. Additionally, the paper discusses the balance between using computational tools and traditional methods to foster a comprehensive understanding of mathematical principles. The implications for future educational practices and the development of computational thinking are also considered, emphasizing the potential benefits and limitations of technology in educational settings.

Keywords: polynomial functions; Python-based tools; secondary mathematics education; computational thinking; interactive learning

1. Introduction

In the realm of secondary mathematics education, understanding and solving polynomial functions are essential for students' mathematical development. Polynomial functions not only form the basis for more advanced studies in mathematics and sciences but also foster critical thinking and problem-solving skills [1]. The ability to manipulate and solve equations involving polynomial expressions is crucial, as these skills are applicable in various real-world contexts, from physics to economics, providing a foundational toolset for understanding complex relationships [2].

Despite the availability of numerous methods for solving polynomial functions, such as symbolic computation and numerical approaches, these techniques often present significant challenges for secondary students [3]. Traditional methods can be too abstract or complex, sometimes involving advanced mathematical concepts beyond the secondary curriculum, which may hinder students' comprehension and engagement [2]. The gap between theoretical methodologies and practical application can hinder effective learning and engagement, particularly for students who struggle with abstract mathematical concepts.

Moreover, classroom environments frequently lack the tools necessary to provide individualized instruction tailored to varying student needs. The traditional one-size-fitsall approach often fails to accommodate learners who require more interactive or visual methods to grasp mathematical concepts. The increasing demand for incorporating digital literacy and computational thinking into the curriculum further emphasizes the need for innovative teaching strategies. Python, a widely used programming language in both academic and professional fields, offers an ideal platform for developing such educational tools. Its readability, extensive library support, and ability to handle complex mathematical operations make it accessible even for beginners.

Therefore, this paper explores how integrating Python-based computational tools into secondary mathematics education can serve as a bridge between complex polynomial theory and student comprehension. It posits that such an approach supports diverse learning styles and promotes active engagement, while also reinforcing foundational skills and introducing students to essential elements of computational thinking. By merging theoretical instruction with practical, technology-driven tools, educators can create a more inclusive, effective, and forward-thinking educational environment.

2. Innovations in Teaching Polynomial Functions: Python-Based Solutions

Addressing the complexities associated with traditional methods of solving polynomial functions, recent educational innovations have leveraged technology to enhance learning and engagement. A pioneering approach involves the use of Python-based software tools designed to simplify the process of solving polynomial functions. These tools allow students to input polynomial equations and receive immediate solutions, thereby facilitating a better understanding of the concepts and providing a practical way to verify their solutions [3]. Such technology not only makes the subject matter more accessible but also aligns with modern educational practices that emphasize interactive and student-centered learning.

3. Integration of Python Tools in Educational Settings

The integration of Python-based tools into the mathematics curriculum offers twofold benefits. Firstly, it provides a straightforward means for students to engage with complex polynomial equations in a controlled, error-forgiving environment. Secondly, it enables educators to focus on teaching the underlying principles and applications of polynomial functions rather than the mechanical process of solving them, thus saving valuable instructional time and enhancing the overall learning experience [3]. This approach not only aids in demystifying advanced mathematical topics but also aligns with contemporary educational goals of fostering computational thinking — a skill increasingly recognized as essential in the digital age.

4. Polynomial Root-Finding Algorithm

In the field of numerical analysis, the problem of finding roots of polynomial functions is of significant importance due to its widespread applications across various scientific disciplines. This document outlines an algorithm implemented in Python, designed to find roots of a polynomial defined by user-inputted coefficients and powers. The procedure (as shown in Figure 1) efficiently combines user interaction, validation, and numerical methods to facilitate the calculation of polynomial roots.

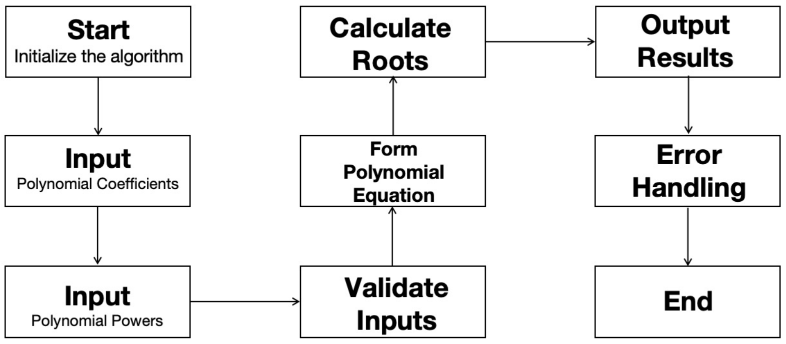

Figure 1. The Procedure of Coding Design.

4.1. Algorithm Description

4.1.1. Step 1: User Input for Polynomial Terms

The algorithm initiates by requesting the user to specify the number of terms in the polynomial, with a limit of 10 terms to ensure manageable complexity. This limitation ensures manageability while maintaining the flexibility to handle a variety of polynomial forms. The number of terms is crucial as it dictates the subsequent input requirements and the complexity of the polynomial being evaluated.

```
num_terms = int (input ("Enter the number of terms (up to 10): "))
```

The validity of the entered number of terms is checked immediately. If the input falls outside the acceptable range (1 to 10), the algorithm provides a feedback message and prompts the user to re-enter the correct input.

4.1.2. Step 2: Capturing Polynomial Powers and Coefficients

The user is then asked to enter the powers for each term of the polynomial. These powers must be non-negative integers and entered in descending order, a requirement that is essential for the correct formulation of the polynomial equation. This step is critical as it defines the structure of the polynomial function.

```
powers = []
print ("Enter the powers of each term (separated by spaces):")
powers_input = input (). split ()
```

After collecting the powers, a similar process is followed, where the user is prompted to input the coefficients for each term. These coefficients are real numbers, which influence the polynomial's behavior and its root locations. Both powers and coefficients are validated for their length and type to ensure they match the expected format and values.

4.1.3. Step 3: Polynomial Construction and Root Calculation

Once the input data are validated and accepted, the algorithm constructs the polynomial function. This construction involves creating a list where the index represents the power of x, and the value at that index represents the coefficient corresponding to that power.

The roots of the polynomial are calculated using Newton's method, a powerful technique for approximating the roots (or zeroes) of a real-valued function.

Newton's method is applied iteratively to find roots, starting from an initial guess provided by the user or generated by the algorithm. For each term in the polynomial, the algorithm attempts to find a root through a series of iterations, refining the guess based on the function's derivative.

4.1.1. Output

The algorithm outputs the roots of the polynomial, each rounded to two decimal places for precision. This output is the culmination of the root-finding process and provides the user with the specific points where the polynomial function intersects the x-axis.

This Python-based root-finding algorithm for polynomials is notable for its user-friendly approach, robust validation, and efficient numerical technique. By allowing users to define the polynomial dynamically and applying Newton's method, it offers a practical and educational tool suitable for academic purposes and practical applications in fields requiring polynomial root solutions.

5. A Four-Term Example for Solving Quadratic Polynomials

To effectively utilize the Python-based polynomial root-finding tool highlighted in our research, users initiate the process by running the get_polynomial_roots function within a Python environment. This begins with the program asking users to specify the number of terms for the polynomial, which is capped at ten.

As shown in Figure 2, inputting 4 indicates that the polynomial will consist of four terms. Users then provide the powers for each term in descending order, for example, three, two, one, and zero, outlining the structure of the polynomial. Following this, users are prompted to enter the coefficients corresponding to these powers. In the given example, these coefficients would be four, two, three, and negative five, which completes the construction of the polynomial equation. The algorithm constructs this polynomial and utilizes Newton's method to calculate the roots, displaying outputs such as zero point seventy-five repeated four times, reflecting the calculated roots with precision generally rounded to two decimal places. It is crucial for users to ensure the numbers of powers and coefficients match accurately and that all inputs conform to the required formats and data types. This method provides an efficient solution to finding roots while enhancing the user's understanding of polynomial structures and computational solving techniques, serving as an educational tool that bridges theoretical concepts with practical application in mathematical learning.

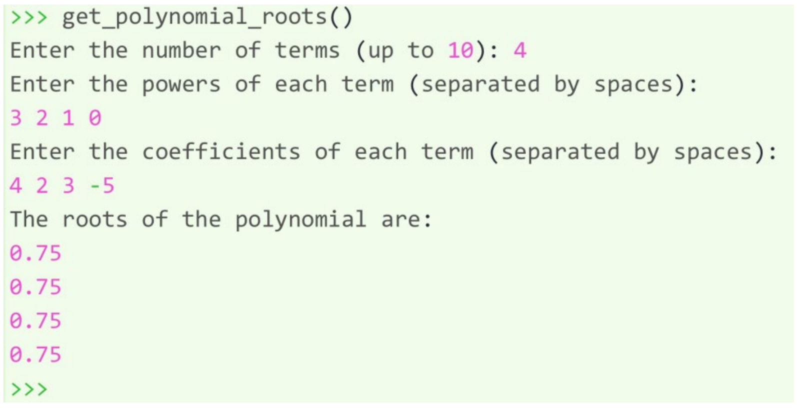
```
>>> get_polynomial_roots()
Enter the number of terms (up to 10): 4
Enter the powers of each term (separated by spaces):
3 2 1 0
Enter the coefficients of each term (separated by spaces):
4 2 3 -5
The roots of the polynomial are:
0.75
0.75
0.75
0.75
>>>
```

Figure 2. An example of using the code.

6. Discussion

The integration of Python-based computational tools in secondary mathematics education enhances student engagement with polynomial functions. Recent advancements in numerical methods have transformed educational paradigms, as demonstrated by Duff, Leykin, and Rodriguez, who highlight the flexibility of Python for personalized learning [3]. Bates et al. emphasize the importance of numerical software, noting its accessibility and ability to handle complex computations essential for understanding polynomials [4]. Santamaría and Woodroofe further illustrate how computational tools simplify complex algebraic structures through interactive modules [5]. Additionally, Breiding and Timme discuss how programming environments like Julia are becoming integral to teaching advanced mathematics, fostering deeper mathematical understanding [6]. However, the over-reliance on technology raises concerns about students neglecting traditional problem-solving skills, as noted by Blum et al., and the risk of undermining rigorous mathematical thinking if students do not understand the underlying algorithms [7-9]. In conclusion, while Python-based tools enhance learning, balancing them with traditional methods.

7. Conclusion

The integration of Python-based computational tools into the teaching of polynomial functions in secondary education represents a transformative advancement in mathematical pedagogy. By streamlining complex problem-solving processes, these tools not only facilitate a deeper understanding of polynomial functions but also enhance student engagement through interactive learning environments. The ability to immediately compute and verify solutions allows students to explore mathematical concepts dynamically, reinforcing theoretical knowledge through practical application. This approach has proven to be invaluable in bridging the gap between abstract mathematical theories and their practical implications, making learning more accessible and enjoyable for students.

Furthermore, these tools offer a gateway to computational thinking—a skill increasingly necessary in modern education and the workforce. Students exposed to Python gain a dual advantage: they not only master the mathematical content but also develop coding proficiency and logical reasoning skills. This interdisciplinary learning supports long-term academic and career readiness, especially in STEM-related fields.

However, the adoption of computational tools should not replace traditional teaching methods entirely. It is crucial to maintain a balance where students develop a solid understanding of underlying mathematical principles without becoming overly reliant on automation. Educators must be trained to integrate these tools thoughtfully, ensuring that technology complements rather than overshadows fundamental skills.

Looking ahead, the continued refinement of educational technology and its integration into curricula offers exciting possibilities for reshaping mathematics education. Future research should explore long-term outcomes of such pedagogical approaches, including how they affect student performance, confidence, and interest in mathematics. Ultimately, the use of Python-based tools in polynomial function instruction highlights a broader shift toward a more dynamic, interactive, and inclusive learning experience, one that prepares students not just to solve equations, but to think critically and creatively in a digitally evolving world.

References

1. M. Živković, S. Pellizzoni, I. C. Mammarella, and M. C. Passolunghi, "Executive functions, math anxiety and math performance in middle school students," *Br. J. Dev. Psychol.*, vol. 40, no. 3, pp. 438–452, 2022, doi: 10.1111/bjdp.12412.
2. S. B. Belhaouari, M. H. F. Hijab, and Z. Oflaz, "Matrix approach to solve polynomial equations," *Results Appl. Math.*, vol. 18, p. 100368, 2023, doi: 10.1016/j.rinam.2023.100368.
3. T. Duff, A. Leykin, and J. I. Rodriguez, "U-generation: Solving systems of polynomials equation-by-equation," *Numer. Algorithms*, vol. 95, no. 2, pp. 813–838, 2024, doi: 10.1007/s11075-023-01590-1.
4. D. J. Bates, A. J. Sommese, J. D. Hauenstein, and C. W. Wampler, *Numerically Solving Polynomial Systems with Bertini*, Philadelphia, PA, USA: SIAM, 2013. ISBN: 9781611972696.
5. A. Santamaría-Galvis and R. Woodroofe, "Shellings from relative shellings, with an application to NP-completeness," *Discrete Comput. Geom.*, vol. 66, no. 2, pp. 792–807, 2021, doi: 10.1007/s00454-020-00273-1.
6. P. Breiding and S. Timme, "HomotopyContinuation.jl: A package for homotopy continuation in Julia," in *Mathematical Software—ICMS 2018: Proc. 6th Int. Conf.*, South Bend, IN, USA, Jul. 24–27, 2018, pp. 458–465, Springer. ISBN: 9780387982816.
7. L. Blum, F. Cucker, M. Shub, and S. Smale, *Complexity and Real Computation*, New York, NY, USA: Springer, 2012, doi: 10.1007/978-1-4612-0701-6.

8. J. Pérez, E. Dapena, J. Aguilar, and G. Carrillo, "Reinforcement learning for estimating student proficiency in math word problems," in *Proc. 2022 XVII Latin Amer. Conf. Learn. Technol. (LACLO)*, Oct. 2022, pp. 01–06, IEEE, doi: 10.1109/LACLO56648.2022.10013399.
9. A. N. Jensen, "Tropical homotopy continuation," arXiv preprint arXiv:1601.02818, 2016, doi: 10.48550/arXiv.1601.02818.

Disclaimer/Publisher's Note: The views, opinions, and data expressed in all publications are solely those of the individual author(s) and contributor(s) and do not necessarily reflect the views of PAP and/or the editor(s). PAP and/or the editor(s) disclaim any responsibility for any injury to individuals or damage to property arising from the ideas, methods, instructions, or products mentioned in the content.