Recursive Polynomial Machine (RPM) Method

Immense Raj Subedi
Department of Mathematics, Gaindakot Namuna
immenserajsubedi@gmail.com

4-19-2025

Abstract

This paper introduces a new method for generating polynomials based on recursive difference analysis, termed the Recursive Polynomial Machine (RPM). RPM efficiently derives the exact closed-form polynomial by recursively analyzing finite differences until stabilization, ensuring minimal degree. RPM halts when the sequence reaches a steady difference, ensuring minimal polynomial degree.

1 Introduction

Given a sequence of values $F(1), F(2), F(3)$, ..., we propose a method to construct a polynomial $P(n)$ that generates the sequence. The key observation is that a sequence can be represented by a polynomial whose differences, taken recursively, eventually stabilize at a constant value.

2 General Formula for RPM

For a sequence that follows a polynomial pattern, the general form for the Recursive Polynomial Machine is:

$$F(n) = F(1) + (n-1)a + \frac{(n-1)(n-2)}{2!}b + \frac{(n-1)(n-2)(n-3)}{3!}c + \cdots$$

Where:

- $F(1)$ is the first term of the sequence,
- a is the first-order difference, calculated as the difference between consecutive terms, i.e. $a = \Delta_1 = F(2) - F(1)$,
- b is the second-order difference, calculated as the difference between consecutive first-order differences, i.e. $b = \Delta_1^2 = \Delta_2 - \Delta_1$,
- c is the third-order difference, and so on for higher-order differences.

The process of adding terms continues until a steady difference is found. When the difference becomes constant (i.e., does not change further), we stop adding higher-order terms, as this indicates that the polynomial degree has been reached. This ensures that we find the minimal degree polynomial necessary to represent the sequence.

This general approach can be extended to sequences of any degree, with the order of the differences increasing accordingly.

3 Recursive Polynomial Summation Formula

An important recursive property of the RPM method emerges when we view the construction of the polynomial values using summations over lower-degree difference polynomials.

Let $f(n)$ be a polynomial of degree n. Then,

$$f(n) = f(1) + \sum_{k=1}^{n-1} P_{n-1}(k)$$

where $P_{n-1}(k)$ is the polynomial derived from the $(n-1)^{\text{th}}$ difference layer, evaluated at k.

Proof. Using the forward difference operator Δ, we define:

$$\Delta f(k) = f(k+1) - f(k)$$

Successive applications yield:

$$\Delta^2 f(k) = \Delta(\Delta f(k)) = f(k+2) - 2f(k+1) + f(k)$$

and so on.

For a polynomial of degree n, the n^{th} difference $\Delta^n f(k)$ is constant. Each difference reduces the degree of the polynomial by one.

To compute $f(n)$ from $f(1)$, we incrementally add the first differences:

$$f(n) = f(1) + \sum_{k=1}^{n-1} \Delta f(k)$$

Since $\Delta f(k)$ represents a polynomial of degree $n-1$, we denote it as $P_{n-1}(k)$. Substituting, we get:

$$f(n) = f(1) + \sum_{k=1}^{n-1} P_{n-1}(k)$$

□

This identity highlights the recursive structure of the RPM method. It allows reconstruction of the n^{th} term of a degree-n polynomial by summing evaluations of a degree-$n-1$ polynomial derived from the finite difference table. This formulation not only demonstrates the symbolic nature of the RPM approach but also provides a pathway for recursive polynomial generation.

4 Summation Formula Derived from RPM

Given a polynomial sequence generated by the Recursive Polynomial Machine method, we can derive the summation formula for the first n terms as follows:

The sum $S(n)$ of the first n terms of a polynomial sequence is:

$$S(n) = (n-1)F(1) + \frac{(n-1)(n-2)}{2}a + \frac{(n-1)(n-2)(n-3)}{6}d + \cdots$$

where:

- $F(1)$ is the first term of the sequence
- $a = \Delta_1 = F(2) - F(1)$ is the first-order difference

- $d = \Delta_1^2 = \Delta_2 - \Delta_1$ is the second-order difference
- Subsequent terms involve higher-order differences until stabilization

Proof. The formula follows from:

1. The RPM expansion $F(k) = F(1) + (k-1)a + \frac{(k-1)(k-2)}{2}d + \cdots$
2. Summing from $k = 1$ to n: $\sum_{k=1}^{n} F(k)$
3. Applying combinatorial identities for falling factorials
4. Terminating when differences become constant

□

For $F(k) = k^2$ with:

- $F(1) = 1$
- $a = 4 - 1 = 3$
- $d = (9-4) - (4-1) = 2$

The summation becomes:

$$S(n) = (n-1)(1) + \frac{(n-1)(n-2)}{2}(3) + \frac{(n-1)(n-2)(n-3)}{6}(2)$$
$$= \frac{n(n+1)(2n+1)}{6} \quad \text{(after simplification)}$$

Key properties of the RPM summation:

- Exact for polynomial sequences
- Self-terminating when differences stabilize
- Coefficients are binomial coefficients with falling factorials
- Generalizes standard summation formulas

5.Uses On some Function:-

A. RPM on a Quadratic Function with Non-Integer Inputs

Let the function be:

$$F(x) = x^2 + x$$

We evaluate the function at non-integer values to generate a sequence:

$$F(0.3) = 0.3^2 + 0.3 = 0.39,$$

$$F(1.3) = 1.3^2 + 1.3 = 2.99,$$

$$F(2.3) = 2.3^2 + 2.3 = 7.59,$$

$$F(3.3) = 3.3^2 + 3.3 = 14.19.$$

So, the sequence is:

$$0.39,\ 2.99,\ 7.59,\ 14.19$$

Step 1: First-Order Differences

$$\Delta_1 = 2.99 - 0.39 = 2.6,$$

$$\Delta_2 = 7.59 - 2.99 = 4.6,$$

$$\Delta_3 = 14.19 - 7.59 = 6.6.$$

Step 2: Second-Order Differences

$$\Delta_1^2 = 4.6 - 2.6 = 2,$$

$$\Delta_2^2 = 6.6 - 4.6 = 2.$$

Since the second-order differences are constant, the sequence follows a quadratic pattern.

Step 3: Applying RPM

Using the RPM formula:

$$F(n) = F(0.3) + (n - 0.3) \cdot a + \frac{(n - 0.3)(n - 1.3)}{2} \cdot d$$

Where:

$$F(0.3) = 0.39, \quad a = 2.6, \quad d = 2$$

Substitute and expand:

$$F(n) = 0.39 + 2.6(n - 0.3) + (n - 0.3)(n - 1.3)$$

Step 4: Simplification

Expand the terms:

$$2.6(n - 0.3) = 2.6n - 0.78$$

$$(n - 0.3)(n - 1.3) = n^2 - 1.6n + 0.39$$

Now substitute:

$$F(n) = 0.39 + 2.6n - 0.78 + n^2 - 1.6n + 0.39$$

Combine like terms:

$$F(n) = n^2 + (2.6n - 1.6n) + (0.39 - 0.78 + 0.39)$$

$$F(n) = n^2 + n$$

Conclusion

Using RPM on the sequence generated by evaluating $x^2 + x$ at $x = 0.3,1.3,2.3,3.3$, we successfully recovered the original polynomial:

$$F(x) = x^2 + x$$

This demonstrates that RPM works not only for integer-domain sequences, but also for sequences where the domain increases linearly by a non-integer step. The Numerical Stability Analysis is included in section no 6.

B. Cubic Sequence Example

Consider the cubic sequence:

$$F(1) = 1, \quad F(2) = 8, \quad F(3) = 27, \quad F(4) = 64.$$

First, calculate the first-order differences:

$$\Delta_1 = F(2) - F(1) = 8 - 1 = 7,$$

$$\Delta_2 = F(3) - F(2) = 27 - 8 = 19,$$

$$\Delta_3 = F(4) - F(3) = 64 - 27 = 37.$$

Next, calculate the second-order differences:

$$\Delta_1^2 = \Delta_2 - \Delta_1 = 19 - 7 = 12,$$

$$\Delta_2^2 = \Delta_3 - \Delta_2 = 37 - 19 = 18.$$

Finally, calculate the third-order difference:

$$\Delta_1^3 = \Delta_2^2 - \Delta_1^2 = 18 - 12 = 6.$$

Since the third-order differences are constant, we conclude that the sequence is cubic.

Now, using the RPM formula for cubic sequences, we have:

$$F(n) = F(1) + (n-1) \cdot a + \frac{(n-1)(n-2)}{2} \cdot d + \frac{(n-1)(n-2)(n-3)}{6} \cdot e.$$

Where: - $F(1) = 1$, - $a = 7$, - $d = 12$, - $e = 6$.

Thus, the polynomial becomes:

$$F(n) = 1 + (n-1) \cdot 7 + \frac{(n-1)(n-2)}{2} \cdot 12 + \frac{(n-1)(n-2)(n-3)}{6} \cdot 6.$$

Simplifying:

$$F(n) = 1 + 7(n-1) + 6(n-1)(n-2) + (n-1)(n-2)(n-3).$$

We now verify the polynomial with the given values: - For $n = 1$:

$$F(1) = 1 + 7(1-1) + 6(1-1)(1-2) + (1-1)(1-2)(1-3) = 1.$$

- For $n = 2$:

$$F(2) = 1 + 7(2-1) + 6(2-1)(2-2) + (2-1)(2-2)(2-3) = 8.$$

- For $n = 3$:

$$F(3) = 1 + 7(3-1) + 6(3-1)(3-2) + (3-1)(3-2)(3-3) = 27.$$

- For $n = 4$:

$$F(4) = 1 + 7(4-1) + 6(4-1)(4-2) + (4-1)(4-2)(4-3) = 64.$$

Thus, the polynomial correctly reproduces the sequence.

C. Fibonacci Sequence Approximation Using RPM

The Fibonacci sequence is defined recursively as:

$$F(n) = F(n-1) + F(n-2), \quad \text{with} \quad F(0) = 0, \quad F(1) = 1.$$

Although this sequence is not inherently polynomial, we can use the Recursive Polynomial Machine (RPM) to fit a polynomial to a finite number of Fibonacci values. This gives an exact match for the selected values, though the pattern does not extend beyond the chosen domain.

Using the First 6 Fibonacci Numbers

We use:

$$F(0) = 0, \quad F(1) = 1, \quad F(2) = 1, \quad F(3) = 2, \quad F(4) = 3, \quad F(5) = 5.$$

Step 1: Construct the Difference Table

n	$F(n)$	Δ_1	Δ^2	Δ^3	Δ^4	Δ^5
0	0	1	-1	2	-3	5
1	1	0	1	-1	2	
2	1	1	0	1		
3	2	1	1			
4	3	2				
5	5					

Step 2: Apply the RPM Formula

The general RPM form for a degree-5 polynomial is:

$$F(n) = F(0) + n\Delta_1(0) + \frac{n(n-1)}{2}\Delta^2(0) + \frac{n(n-1)(n-2)}{6}\Delta^3(0)$$
$$+ \frac{n(n-1)(n-2)(n-3)}{24}\Delta^4(0) + \frac{n(n-1)(n-2)(n-3)(n-4)}{120}\Delta^5(0)$$

Where:

- $F(0) = 0$

- $\Delta_1(0) = 1$
- $\Delta^2(0) = -1$
- $\Delta^3(0) = 2$
- $\Delta^4(0) = -3$
- $\Delta^5(0) = 5$

Substituting the values:

$$F(n) = 0 + n \cdot 1 + \frac{n(n-1)}{2} \cdot (-1) + \frac{n(n-1)(n-2)}{6} \cdot 2 + \frac{n(n-1)(n-2)(n-3)}{24} \cdot (-3) + \frac{n(n-1)(n-2)(n-3)(n-4)}{120} \cdot 5$$

Simplifying:

$$F(n) = n - \frac{n(n-1)}{2} + \frac{n(n-1)(n-2)}{3} - \frac{n(n-1)(n-2)(n-3)}{8} + \frac{n(n-1)(n-2)(n-3)(n-4)}{24}$$

Step 3: Verify Values

- For $n = 0$:

$$F(0) = 0 - 0 + 0 - 0 + 0 = 0$$

- For $n = 1$:

$$F(1) = 1 - 0 + 0 - 0 + 0 = 1$$

- For $n = 2$:

$$F(2) = 2 - 1 + 0 - 0 + 0 = 1$$

- For $n = 3$:

$$F(3) = 3 - 3 + 2 - 0 + 0 = 2$$

- For $n = 4$:

$$F(4) = 4 - 6 + 8 - 3 + 0 = 3$$

- For $n = 5$:

$$F(5) = 5 - 10 + 20 - 15 + 5 = 5$$

Step 4: Final Polynomial Form

The expanded form is:

$$F(n) = \frac{1}{24}n^5 - \frac{1}{4}n^4 + \frac{23}{24}n^3 - \frac{3}{4}n^2 + \frac{13}{12}n$$

Remark

This polynomial exactly matches $F(0)$ through $F(5)$ but diverges for $n \geq 6$, demonstrating that RPM provides a local polynomial approximation for non-polynomial sequences. The approximation quality degrades rapidly outside the fitted range.

6.Numerical Stability Analysis

When applying the Recursive Polynomial Machine (RPM) to sequences generated from floating-point inputs (e.g., x=0.3,1.3,2.3, …), it is important to consider the effects of numerical precision and stability, particularly when using small or irrational step sizes.

Precision and Floating-Point Errors:-

In floating-point arithmetic, operations such as subtraction can introduce precision loss—especially when the operands are nearly equal. Within RPM, this may occur during:

- Computation of higher-order differences,
- Expansion and simplification of expressions involving decimal inputs, and
- Subtraction steps within difference tables.

Nevertheless, RPM maintains numerical stability in most practical scenarios due to the following:

- The difference table only amplifies rounding errors if the input sequence contains substantial noise or poorly conditioned values.
- For standard step sizes (e.g., 0.1, 0.3), double-precision floating-point arithmetic (IEEE 754) typically results in negligible error—on the order of $O(10^{-15})$.
- When symbolic computation or extended precision is used, the final polynomial (e.g.,$F(x) = x^2 + x$,) is recovered exactly, even from decimal-valued sequences.

Example Revisited

In the previous example with:

$$F(x) = x^2 + x, \quad x = 0.3,1.3,2.3,3.3,$$

The RPM correctly reconstructed:

$$F(n) = n^2 + n$$

Despite working with decimal input points. This illustrates that the method is numerically stable for well-behaved polynomial sequences.

7.Comparison with Newton's Method

The Recursive Polynomial Machine (RPM) offers an alternative to Newton's method for polynomial construction. Below is a comparison of the two approaches:

- **Degree of Polynomial**:
    - **Newton's Method**: Generates a polynomial of degree $n-1$ for n data points, often resulting in high-degree polynomials even for small datasets. This can lead to overfitting.
    - **RPM**: Stops adding higher-order terms once a steady difference is found, ensuring the minimal degree polynomial necessary to represent the sequence.
- **Efficiency**:
    - **Newton's Method**: Requires calculation of divided differences for all data points, which can be computationally expensive for large datasets.
    - **RPM**: Uses a simpler difference calculation and halts when steady differences are found, reducing unnecessary computations and improving efficiency.
- **Flexibility**:
    - **Newton's Method**: Constructs a polynomial incrementally based on all data points, but doesn't provide insight into the minimal degree of the polynomial.
    - **RPM**: Halts once steady differences are found, adapting to the sequence's complexity and ensuring a minimal-degree polynomial.
- **Overfitting**:
    - **Newton's Method**: May lead to overfitting, especially for small datasets or when data points are spread out. The high-degree polynomial may not generalize well.
    - **RPM**: Prevents overfitting by constructing a polynomial of minimal degree that fits the sequence.
- **Application Scope**:
    - **Newton's Method**: Commonly used for interpolation, particularly when an exact fit for all data points is required.
    - **RPM**: A general method for generating polynomials from sequences, with a focus on minimizing the polynomial degree while fitting the data.
- **Ease of Interpretation**:

- **Newton's Method**: The polynomial may become complex and difficult to interpret when the degree increases, especially with many data points.
- **RPM**: The recursive structure of RPM makes it more interpretable, as each term corresponds to a specific difference in the sequence.

Comparison Conclusion: While both methods aim to construct a polynomial from data points, RPM offers the advantage of generating a minimal-degree polynomial that fits the sequence efficiently and avoids overfitting. It is a more flexible and interpretable method compared to Newton's method, especially for sequences where a simple polynomial suffices.

8.Recommendations

To ensure maximum numerical accuracy in practical implementations of RPM:

- Use symbolic computation or high-precision libraries where possible.
- Avoid inputs with extremely small step differences unless higher precision is used.
- Round difference values to a tolerable threshold (e.g., 10^{-10}) when checking for steady differences.

9.Conclusion

RPM remains numerically stable for typical applications, even with non-integer domains. Minor floating-point inaccuracies may appear, but the overall symbolic structure of the polynomial can still be recovered accurately.