Polynomial Methods for Ensuring Data Integrity in Financial Systems

Ignacio Brasca

July 30, 2024

1 Introduction

In any sufficiently complex system, such as any vast financial system with any practical user base, ensuring data integrity across K data points is crucial.

Furthermore, to satisfy correctness during usage, we should ensure that k_n data points are verified and correct for consumption while K is correct as a whole, without any discrepancies in expected values.

Values falling out of valid ranges can cause distress among consumers of this data, leading to questions directed at platform designers, resulting in wasted time trying to recover original data points used, mitigating data loss, and validating inputs across I indicators where I is the number of total indicators used across the platform. (An indicator is a set of k_n configured in a way that an operation produces a result consumed by the end-consumer.)

To address these issues, we propose implementing an algorithm that relies on the classical Lagrange Interpolation to help maintain correctness in our set of I by using polynomials through known data points.

2 Background

Indicators (I_n) play a fundamental role in any financial system where, for each specific I_n, a set $\{k_1, k_2, \dots, k_n\}$ could be in use.

$$I_n = \sum_{i=0}^{k_n} f(k_i) \tag{1}$$

where $f(k_i)$ is a function that takes a data point and produces a result. This is called an *operation* and is used to calculate any measure required to obtain I_n.

An indicator, in conclusion, is a function that takes a set of data points and produces a result used in the calculation of other indicators or as an individual indicator ready for consumption.

Polynomials

The usage of polynomials to maintain consistency is well-discussed in the literature. Algorithms such as Reed-Solomon codes, used in error detection and correction, rely on polynomials to ensure data integrity [2], [6], [4].

This, in addition to the simplicity of polynomial representation, makes it a good candidate for ensuring data integrity.

Additionally, polynomials are key in solving complex problems in mathematics by breaking them down into simpler functions that can be recombined to form the original problem. Given any polynomial $P(x)$, there exists a unique function $y = \phi(t)$ that satisfies the differential equation:

$$y' + p(t)y = g(t), \quad y(t_0) = y \tag{2}$$

This guarantees the existence and uniqueness of the polynomial for given points [10].

Knowing **existence** and **uniqueness** from a set of points, we can rely on that to understand how we can use this to ensure the integrity of our data points.

Shamir Secret Sharing [12] is another example of a cryptographic algorithm that relies on polynomial interpolation to recover a secret from a set of shares, leveraging the same properties of polynomials described later in this document.

Lagrange Interpolation

Lagrange Interpolation [1] is a method used to find a polynomial $P(x)$ that passes through a set of given points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$. The polynomial $P(x)$ is constructed as a linear combination of basis polynomials:

$$P(x) = \sum_{i=0}^{n} y_i L_i(x) \tag{3}$$

where $L_i(x)$ are the Lagrange basis polynomials defined as:

$$L_i(x) = \prod_{\substack{0 \leq j \leq n \\ j \neq i}} \frac{x - x_j}{x_i - x_j} \tag{4}$$

Lagrange Interpolation: An example

Consider the points $(1, 2)$, $(2, 3)$, and $(3, 5)$. Using Lagrange Interpolation, the basis polynomials are:

$$L_0(x) = \frac{(x-2)(x-3)}{(1-2)(1-3)} = \frac{(x-2)(x-3)}{2} \tag{5}$$

$$L_1(x) = \frac{(x-1)(x-3)}{(2-1)(2-3)} = -(x-1)(x-3) \tag{6}$$

$$L_2(x) = \frac{(x-1)(x-2)}{(3-1)(3-2)} = \frac{(x-1)(x-2)}{2} \tag{7}$$

The interpolating polynomial $P(x)$ is given by:

$$P(x) = \sum_{i=0}^{2} y_i L_i(x) \tag{8}$$

Simplifying, we obtain:

$$P(x) = \frac{1}{2}x^2 - \frac{1}{2}x + 2 \tag{9}$$

This polynomial passes through the points $(1, 2)$, $(2, 3)$, and $(3, 5)$.

3 Application

The application of polynomials in our context allows us to recover and validate data points, ensuring continuity and uniqueness of the results across all inputs.

We describe a $k_1, k_2, \ldots, k_n$ set of data points used to generate a polynomial from a list of arguments used in the I_n indicator.

Suppose we have a set of data points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ to compute an indicator I_n.

Furthermore, we take a subset of these original data points and assume $[x_0 = 0, x_1 = 1, x_2 = 2, \ldots, x_n = n]$, where the value of $f(x)$ at each x_i corresponds to the original data point's value. For example, $f(0) = X$, $f(1) = Y$, $f(2) = Z$, and so on up to $f(n) = n$.

From there, we can start describing a $f(x)$ polynomial from a set of points first defined by our data explanation from k_n sample data points. After presenting a set of points, already defined, we can start interpolating the polynomial from the set of those same exact points.

Giving us a list of $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, defined across $(0, f(0)), (1, f(1)), \ldots, (n, f(n))$ from which we now have a polynomial that describes the function $f(x)$:

$$f(x) = \sum_{i=0}^{n} y_i L_i(x) \tag{10}$$

where $L_i(x)$ is the Lagrange basis polynomial defined as:

$$L_i(x) = \prod_{\substack{0 \leq j \leq n \\ j \neq i}} \frac{x - x_j}{x_i - x_j} \tag{11}$$

Using $f(x)$, we can now generate a set of m parity blocks that can be used to recover the original data points in case of data loss or corruption (as long as we store those blocks in a different place, see section 5).

Ideally, one can generate as many k (where k is the number of points used in the first place) parity blocks as needed; however, we can generate as many

m blocks as we want and store them across different data storages.

One caveat is that we can reconstruct the original value as long as we have at least k points, where k is the threshold number of points required to reconstruct the polynomial, and n is $deg(P(x))$, which is $n = k - 1$.

This ensures original k data points can be reconstructed from the parity blocks, maintaining data integrity and consistency without the need to store the original data points in multiple locations.

Example: Carbon footprint calculation

One notable application is the calculation of the carbon footprint from the total investments of a portfolio [3]:

$$\text{carbon footprint} = \frac{\text{total scope emissions}}{\text{total value of investments}} \tag{12}$$

Breaking this down, the total scope emissions can be derived from individual companies' emissions. By using Lagrange Interpolation, we can ensure that the calculated carbon footprint is consistent and accurate.

For three companies with emissions:

$$\begin{aligned} \text{Company A:} &\quad \text{300 tonnes} && (13)\\ \text{Company B:} &\quad \text{400 tonnes} && (14)\\ \text{Company C:} &\quad \text{300 tonnes} && (15)\\ \text{Total Value:} &\quad \text{3000 EUR} && (16) \end{aligned}$$

We can define the polynomial $P(x)$ denoted by the set of points: $(1, 300)$, $(2, 400)$, $(3, 300)$, $(4, 3000)$. The Lagrange Interpolation polynomial is:

$$P(x) = \frac{1}{6}x^3 - \frac{3}{2}x^2 + \frac{11}{6}x - 100 \tag{17}$$

By interpolating these values, we can recover the original carbon footprint even if some data points are lost.

4 Concerns and Data Recovery

4.1 Concerns

One primary concern is the possibility of data corruption or loss. To mitigate this, we store data points in multiple locations and use redundancy to recover lost data. The method also ensures that any interpolation is reversible, allowing us to verify the integrity of the data. This redundancy is analogous to techniques used in Reed-Solomon error correction codes, which can correct multiple errors in data blocks [9], [5].

4.2 Data Recovery Example

To demonstrate data recovery, consider the earlier example of the carbon footprint calculation. If we lose some data points, we can use the remaining points and the interpolating polynomial to reconstruct the missing values. This is done using the Lagrange Interpolation formula and the stored data points.

5 Storing Data Points

To further enhance data integrity, we store data points in a secondary data storage. This ensures that even in the case of primary storage failure, the data can be recovered from the secondary storage. The steps include:

1. Prepare data parts for k data points.

2. Construct the polynomial using Lagrange Interpolation.

3. Sample additional points (parity blocks) and store them in secondary storage.

4. Use the stored data and parity blocks to recover the original values if needed.

This approach is similar to RAID 6 storage systems, which use Reed-Solomon codes to provide fault tolerance and data recovery capabilities [8], [13].

5.1 Example

```
fn store_data_points(data_points: Vec<f64>) {
    // construct polynomial from data points
    let polynomial = polynomial(&data_points);

    // sample parity blocks
    let parity_blocks =
        sample_parity(&polynomial);

    // store data points and parity blocks
    store_in_primary_storage(&data_points);
    store_in_secondary_storage(&parity_blocks);
}

fn recover_data_points(data_points: Vec<f64>)
     -> Vec<f64> {
    // check if data points exist, if not,
        recover from backup
    let data_points = if !in_primary_storage() {
        retrieve_from_backup()
    } else {
        retrieve_from_primary()
    };

    // construct polynomial from available data
        points
    let polynomial = construct(&data_points);

    polynomial.interpolate(&data_points)
}
```

Listing 1: Pseudo code for storing data points

6 Use Cases in Other Fields

6.1 Medical Imaging

In medical imaging, accurate data reconstruction is crucial. Lagrange Interpolation can be used to fill in missing or corrupted pixel values in MRI and CT scans, ensuring accurate and reliable images for diagnosis [7].

6.2 Climate Modeling

Climate models rely on vast amounts of data from various sources. Lagrange Interpolation can help in interpolating missing data points from temperature, humidity, and other climatic variables, ensuring the models are accurate and robust [2].

6.3 Engineering Design

In engineering design, particularly in finite element analysis, interpolating values at various points in a mesh is essential. Lagrange Interpolation can provide accurate approximations of physical properties across the mesh [6].

7 Secondary Database for Data Recovery

Consider a scenario where we have a critical equation used in financial forecasting:

$$F(t) = a \cdot e^{bt} + c \cdot \sin(dt) \tag{18}$$

The coefficients a, b, c, d are stored in a primary database. In the event of a database failure, we can recover these coefficients using Lagrange Interpolation from a secondary database where interpolated parity blocks are stored. If the primary database fails or is corrupted, the following steps can be used to recover the coefficients:

7.1 Recovery Procedure

Step 1: Check Primary Database Accessibility

First, determine whether the primary database is accessible. This involves checking the network connectivity, server status, and database health. Ensuring the primary database is online and functioning correctly is crucial before proceeding to data retrieval.

Step 2: Retrieve Parity Blocks from Secondary Database

If the primary database is not accessible, initiate the retrieval process for parity blocks stored in the secondary database. These parity blocks are essential for reconstructing the missing data and are typically stored in a fault-tolerant manner.

Step 3: Reconstruct Missing Coefficients Using Lagrange Interpolation

Utilize the retrieved parity blocks and the known data points to perform Lagrange Interpolation. This mathematical technique will help in reconstructing the missing coefficients, ensuring that the recovered data is accurate and reliable.

Step 4: Validate Recovered Coefficients

Finally, validate the reconstructed coefficients by comparing them against known data points. This step is crucial for verifying the accuracy and integrity of the recovered data, ensuring consistency across the system.

8 Conclusion

The technique described here can be used to ensure data integrity across a set of I indicators and maintain consistency in a system where traceability of the I_n in use is critical.

This technique can be applied in any critical system where robustness is mandatory. By relying on the mathematical properties of the Lagrange Interpolation, we can be sure our function will be well-defined in the domain of $\mathbb{R}$.

Storing parity blocks in secondary databases allows us to ensure from them k_{parity} and discard corrupted (or untrusted) data points. These parity blocks enable us to trace back across the domain and recover the set $\{(0, f(0)), (1, f(1)), \ldots, (n, f(n))\}$ data points used to generate the polynomial $f(x)$ in the first place. This remains true even if all the original data points are lost or corrupted, thanks to extrapolation and reliance on the uniqueness and existence of the polynomial $f(x)$ in the domain of $\mathbb{R}$ [11].

This approach provides fault tolerance (up to m parity blocks) and data recovery capabilities. By implementing this technique, we can enhance data integrity and ensure the continuity of critical operations in various domains.

9 Future Work

Future work includes the implementation of the described technique in real-world scenarios and the evaluation of its performance in terms of data recovery, accuracy, and computational efficiency.

10 References

References

[1] Burden, R. L., and Faires, J. D. *Numerical Analysis*, 7th ed. Brooks/Cole, 2001.

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms.* MIT Press, 2009.

[3] ESMA. Esma technical advice: Final report on draft regulatory technical standards, 2023. Carbon Footprint Calculation.

[4] Lidl, R., and Niederreiter, H. *Introduction to Finite Fields and Their Applications.* Cambridge University Press, 1986.

[5] Lin, S., and Costello, D. J. *Error Control Coding: Fundamentals and Applications.* Prentice Hall, 1983.

[6] MacWilliams, F. J., and Sloane, N. J. A. *The Theory of Error-Correcting Codes.* North-Holland, 1977.

[7] McEliece, R. J. *The Theory of Information and Coding.* Addison-Wesley, 1977.

[8] Patterson, D. A., Gibson, G., and Katz, R. H. A case for redundant arrays of inexpensive disks (raid). In *ACM SIGMOD Conference* (1988).

[9] Peterson, W. W., and Weldon, E. J. *Error-Correcting Codes.* MIT Press, 1972.

[10] Rana, I. K. *An Introduction to Measure and Integration.* Springer, 2002.

[11] Rudin, W. *Principles of Mathematical Analysis*, 3rd ed. McGraw-Hill, 1976.

[12] Shamir, A. How to share a secret. *Communications of the ACM 22*, 11 (1979), 612–613.

[13] Sudan, M. Decoding of reed-solomon codes beyond the error-correction bound. *Journal of Complexity 13*, 1 (1997), 180–193.