# Plato: Approximate Analytics over Compressed Time Series with Tight Deterministic Error Guarantees

Chunbin Lin†, Etienne Boursier*, Yannis Papakonstantinou‡
†Amazon AWS, *ENS Paris-Saclay, ‡Amazon AWS & UCSD
†lichunbi@amazon.com, *eboursie@ens-paris-saclay.fr, ‡yannip@amazon.com

## ABSTRACT

Plato provides fast approximate analytics on time series, by precomputing and storing compressed time series. Plato's key novelty is the delivery of *tight deterministic error guarantees* for the linear algebra operators over vectors/time series, the inner product operator and arithmetic operators. Composing them allows for evaluating common statistics, such as correlation and cross-correlation. In the offline processing phase, Plato (i) segments each time series into several disjoint segmentations using known fixed-length or variable-length segmentation algorithms; (ii) compresses each segment by a compression function that is coming from a user-chosen compression function family; and (iii) associates to each segment 1 to 3 precomputed error measures. In the online query processing phase, Plato uses the error measures to compute the error guarantees. Importantly, we identify certain compression function families that lead to theoretically and experimentally higher quality guarantees.

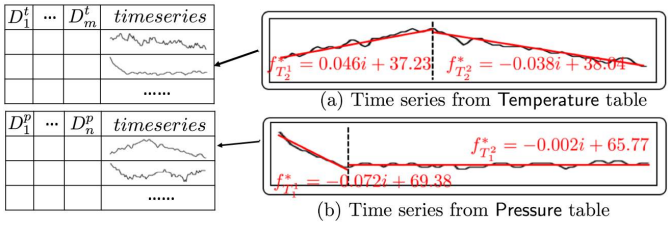
**PVLDB Reference Format:**
Chunbin Lin, Etienne Boursier, and Yannis Papakonstantinou. Approximate Analytics System over Compressed Time Series with Tight Deterministic Error Guarantees. *PVLDB*, 13(7): 1105-1118, 2020.
DOI: https://doi.org/10.14778/3384345.3384357

## 1. INTRODUCTION

Attention to time series analytics is bound to increase in the IoT era as cheap sensors can now deliver vast volumes of many types of measurements. The size of the data is also bound to increase. E.g., an IoT-ready oil drilling rig produces about 8 TB of operational data in one day.[1] One way to solve this problem is to increase the expense in computing and storage in order to catch up. However, in many domains, the data size increase is expected to outpace the increase of computing abilities, thus making this approach unattractive [21, 9]. Another solution is *approximate analytics* over compressed time series.

[1]https://wasabi.com/storage-solutions/internet-of-things/


*Proceedings of the VLDB Endowment,* Vol. 13, No. 7
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3384345.3384357




(a) Time series from Temperature table

(b) Time series from Pressure table

```
SELECT ccorrs.*, approximateAnswer(result), errorGuarantee(result)
FROM (
    SELECT D_1^t, ..., D_m^t, D_1^p, ..., D_n^p,
           TSA('ccorr($1, $2, 5)', t.timeseries, p.timeseries) AS result
    FROM Temperature t, Pressure p
    WHERE cond( t.D_1^t, ..., t.D_m^t, p.D_1^p, ..., p.D_n^p )) AS ccorrs
ORDER BY approximateAnswer(result) DESC
LIMIT 10;
```

(c)*Query*

**Figure 1: Example of SQL query using the time series analytic (TSA) UDF.**

Approximate analytics over historical time series data enables fast computation. For example, consider the database in Figure 1, which has a Temperature table and a Pressure table. Each table contains (i) one timeseries column containing time series data, as a UDT [17] and (ii) several other "dimension" attributes $D$, such as geographic locations and other properties of the sensors that delivered the time series. The Plato SQL query in Figure 1(c) "returns the top-10 temperature/pressure 5-second cross-correlation scores among all the (temperature, pressure) pairs satisfying a (not detailed in the example) condition over the dimension attributes". Notice, the first argument of the Time Series Analytic UDF (TSA) is the expression ccorr($1,$2,5), which does the 5-second cross-correlation of the first argument (t.timeseries) and the second argument (p.timeseries). Computing the accurate cross-correlations would cost more than 10 minutes. However, Plato reduces the runtime to within one second by computing the approximate correlations. It also delivers deterministic error guarantees, which means the error bounds have 100% confidence. In SQL, the result is a JSON string encoding the approximate answer and the error guarantee. The functions approximateAnswer and errorGuarantee extract the respective pieces.

The success of approximate querying on IoT time series data is based on an important beneficial property of the time series data: the points in the sequence of values exhibit *continuity*. For example, a temperature sensor is very unlikely to report a 100 degrees increase within a second.

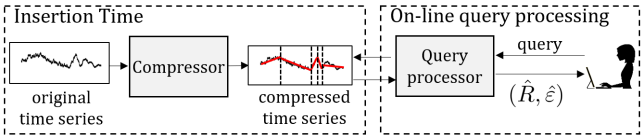



**Figure 2: Plato's Approximate Querying: off-line compression and on-line query processing**

Therefore, in the signal processing and data mining communities [30, 27, 18, 7], time series data is usually modeled and compressed by continuous functions in order to reduce its size. For instance, the Piecewise Aggregate Approximation (PAA) [30] and the Piecewise Linear Representation (PLR) [27] adopt polynomial functions (0-degree in PAA and 1-degree in PLR) to compress the time series; [45] uses Gaussian functions; [55] applies natural logarithmic functions and natural exponential functions to compress time series. We will see that Plato's error guarantee ability is open to any existing time series compression techniques, as long as they compute standard error measures.

**Architecture**. Figure 2 shows the high-level architecture. During insertion time (which is offline processing, for the purposes of query evaluation), the provided time series is compressed. In particular, a *compression function family* (e.g., 2nd-degree polynomials) is chosen by the user. Internally, in a simple version, each time series is segmented (partitioned) first in equal lengths. Then, for each segment the system finds the best *estimation function*, which is the member of the chosen compression function family that best approximates the values in this segment. The compressed database stores the parameters of the estimation function for each segment, which take much less space than the original time series data. In the more sophisticated version, segmentation and estimation are mingled together [34, 28] to achieve better compression. The result is that the time series is partitioned into variable-length segments.

Consequently, given a query $q$ with TSA UDF calls, Plato computes quickly an approximate answer for each TSA call by using the compressed data. Note, the TSAs may combine multiple time series; e.g., a correlation or a cross-correlation.

EXAMPLE 1. *Consider a room temperature time series $T_1$ and an air pressure time series $T_2$ in Figure 1 and consider the* TSA*('Ccorr($T_1, T_2$, 60)', $T_1$, $T_2$) where 'Ccorr($T_1, T_2, 60$)' refers to the 60-seconds cross-correlation of $T_1$ and $T_2$. Both $T_1$ and $T_2$ have* 600 *data points at 1-second resolution, are segmented by variable length segmentation methods and compressed by PLR (1-degree polynomial functions) [19] . The precise answer is* 0.303*. But instead of accessing the* 1200 *(*$600 \times 2$*) original data points, Plato produces the approximate answer* 0.300 *(error is* 0.003*) by accessing just the function parameters (*$-0.072, 69.38$*), (*$-0.002, 65.77$*) for $T_1$ and (*$-0.046, 37.23$*), (*$-0.038, 38.04$*) for $T_2$ in the compressed database.*

The well-known downside of approximate querying is that errors are introduced. When the example's user receives the approximate answer 0.300 she cannot tell how far this answer is from the *true answer*, i.e., the precise answer. The major novelty of Plato is the provision of *tight (i.e., lower bound) deterministic error guarantees* for the answers, even when the time series expressions combine multiple series. In the Example 1, Plato guarantees that the true answer is within $\pm 0.0032$ of the approximate answer 0.300 with 100% confidence. (Indeed, 0.303 is within $\pm 0.0032$ of 0.300.) It produces these guarantees by utilizing *error measures* associated with each segment. The derivation of error guarantees is challenging as each time series is segmented and compressed individually (off-line) before the queries arrive, which results in (i) time series being segmented in misaligned ways, and (ii) different compression functions being utilized in different time series.

The contributions are summarized as follows.

- We deliver tight deterministic error guarantees for the evaluation of the plus, minus, product and inner product vector/timeseries operators over compressed timeseries. These operators, along with the arithmetic operators and the (time)shift operator can be used to compose more complex statistics (eg, cross-correlation); we also show how to compute error guarantees for the compositions. The key challenge is analytics (e.g., correlation and cross-correlation) that combine multiple time series but it is not known in advance which time series may be combined. Thus, each time series has been compressed individually with different compression methods, much before a query arrives. To make the problem harder, time series segmentations are generally misaligned. The guarantees for each TSA operator are tight in the sense that any attempt to create a better (i.e., smaller) error guarantee will fail because we can construct worst-case input time series where the true error is exactly as large as the error guarantee.
- The provided guarantees apply regardless of the specifics of the segmentation and compression function family used during the compression, thus making the provided deterministic error guarantees applicable to any prior work on segment-based compression (eg, variable-sized histograms etc). The only requirements are (i) the common assumption that the estimation function minimizes the Euclidean distance between the actual values and the estimates and (ii) common reconstruction errors for each segment are precomputed.
- We identify broad compression function family groups (namely, the already known Vector Space family (VS) and the presently defined Linear Scalable Family (LSF)) that lead to theoretically and practically high quality guarantees and we provide intuition on the quality difference by pointing out an Amplitude Independence (AI) property. Furthermore, the error guarantees are computed very efficiently, in time proportional to the number of segments.
- We conduct an extensive empirical evaluation on four real-life datasets to evaluate the error guarantees provided by Plato and the importance of the VS and LSF properties on error estimation. The results show that the AI error guarantees are very narrow - thus, practical. Furthermore, we compare to sampling-based approximation and show experimentally that Plato delivers deterministic (100% confidence) error guarantees using fewer data than it takes to produce probabilistic error guarantees with 95% and 99% confidence via sampling.

## 2. TIME SERIES AND EXPRESSIONS

**Time Series** A time series $\boldsymbol{T} = (a, b, [\boldsymbol{T}[a], \boldsymbol{T}[a+1], ..., \boldsymbol{T}[b]])$, $a \in N$, $b \in N$, is a sequence of data points $[\boldsymbol{T}[a], \boldsymbol{T}[a+1], ..., \boldsymbol{T}[b]]$ observed from start time $a$ to end time $b$. Following the assumptions in [43, 10, 56] we assume that time is discrete and the resolution of any two time series is the same. Equivalently, we say $\boldsymbol{T}$ is fully defined in the integer time domain $[a, b]$. We assume a domain $[1, n]$ is the global domain meaning that all the time series are defined within subsets of this domain. When the domain of a time series $\boldsymbol{T}$ is implied by the context, then $\boldsymbol{T}$ can be simplified as $\boldsymbol{T} = [\boldsymbol{T}[a], \boldsymbol{T}[a+1], ..., \boldsymbol{T}[b]]$.

EXAMPLE 2. *Assume the global domain is* $[1, 100]$. *Consider two time series* $\boldsymbol{T}_1 = (1, 5, [61.52, 59.54, 58.64, 59.36, 60.44])$ *and* $\boldsymbol{T}_2 = (3, 6, [1.02, 1.03, 1.02, 1.02])$. *Then* $\boldsymbol{T}_1$ *and* $\boldsymbol{T}_2$ *are fully defined in domains [1,5] and [3,6] respectively.* $\boldsymbol{T}_2[4] = 1.03$ *refers to the* $2^{nd}$ *data point of* $\boldsymbol{T}_2$ *at the 4-th position in the global domain.*

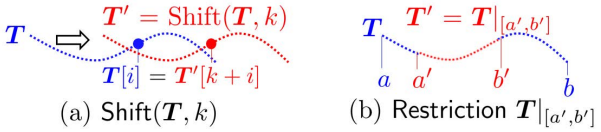
**Time Series Analytic (TSA) Expressions** Table 2 shows the formal definition of the *time series analytic* (called *TSA*) expressions. The TSAs supported are expressions composed of linear algebra operators and arithmetic operators. Typically, the TSA has subexpressions that compose one or more linear algebra operators over multiple time series vectors.

A few examples and intuition behind the operators in Table 2: Constant$(1.6, 3, 5)$ produces $(3, 5, [1.6, 1.6, 1.6])$. Figure 3(a) visualizes the Shift operator. If, say, $\boldsymbol{T} = (1, 3, [1.8, 1.6, 1.6])$, then Shift$(\boldsymbol{T}, 6)$ is $(7, 9, [1.8, 1.6, 1.6])$. Finally, given time series $\boldsymbol{T}_1 = (a_1, b_1, [...])$ and $\boldsymbol{T}_2 = (a_2, b_2, [...])$, notice that the $\boldsymbol{T}_1 \times \boldsymbol{T}_2$ is defined in the intersection of $[a_1, b_1]$ and $[a_2, b_2]$. For example, given $\boldsymbol{T}_1 = (1, 3, [3.3, 3.5, 3.6])$ and $\boldsymbol{T}_2 = (0, 2, [0.9, 1.0, 1.2])$ then $\boldsymbol{T_1} \times \boldsymbol{T_2} = (1, 2, [3.3, 4.2])$. Similarly, we define $\boldsymbol{T_1} + \boldsymbol{T_2}$ and $\boldsymbol{T_1} - \boldsymbol{T_2}$.

When the bounds $a$ and $b$ of a $Sum(\boldsymbol{T}, a, b)$ are implied from the context, we simplify $Sum(\boldsymbol{T}, a, b)$ to $Sum(\boldsymbol{T})$. Table 1 lists several example TSAs for common statistics.

**Scope and Limitations of TSA expressions**. The time series analytic expressions compose the vector operators ($+$, $-$, $\times$, Shift), the arithmetic operators, the aggregation operator Sum that turns its input vector into a scalar, and the Constant operator that turns its input scalar into a vector. As such, Plato queries can express statistics that involve one time series (e.g., average, variance, and n-th moment) and statistics that involve multiple time series, such as correlation and cross-correlation; see Table 1.

Note, the generality of TSAs is not to imply that any vector-based analysis can be expressed just with a TSA expression. Rather, there are forms of analysis where the TSA expressions can be building blocks of a larger computation. One major category of such analytics is cases (e.g., Kalman filters) where an algorithm loops and produces, in each round, a vector $\boldsymbol{T}_k$ by evaluating a TSA expression that uses the vector $\boldsymbol{T}_{k-1}$ computed in the previous round. While the present work trivially allows for the propagation of errors from round-to-round, it is a topic of future work whether the error bound tightness claim will carry over the loop. Another important category of analytics that are not expressible just with TSA is anomaly detection and, generally, analytics that would require extra operators on top of the basic vector/timeseries and arithmetic of Plato.



(a) Shift$(\boldsymbol{T}, k)$ (b) Restriction $\boldsymbol{T}|_{[a',b']}$

**Figure 3: Time series *Shift* and *Restriction* operators.**

## 3. INTERNAL, COMPRESSED TIME SERIES REPRESENTATION

When a user inserts a time series into the database, Plato physically stores the *compressed time series representation* instead of the raw time series. More precisely, the user provides (i) a time series $T$, (ii) a choice between fixed-length segmentation algorithm [14, 20] and variable-length/window-based segmentation algorithm [34, 28, 29, 5, 22, 39] , and (iii) the identifier of a compression function family (e.g., Polynomial function family).

During offline processing, i.e., before the queries are known, Plato uses the chosen segmentation algorithm and the chosen compression function family to partition $T$ into a list of disjoint segments $T^1, ..., T^n$. For each segment $T^i = (a, b, [T^i[a], ..., T^i[b]])$, instead of storing original data points $[T^i[a], ..., T^i[b]]$, Plato stores a *compressed segment representation* $\tilde{T^i} = (a, b, \tilde{f_T^*}, \Phi(T))$, where $a$ is the start position, $b$ is the end position, $\tilde{f_T^*}$ is the function representation (at the storage level) of the estimation function $f_T^*$ chosen from the identified function family and $\Phi(T)$ is a set of (one to three, depending on the function family) *error measures*.

Overall, for a time series $T$, Plato physically stores (i) the list $L_T$=$(\tilde{T^1}, ..., \tilde{T^n})$, and (ii) one token (which can simply be an integer) as the compression function family identifier.

Next, we introduce the segmentation methods employed by Plato (Section 3.1), the selection of the estimation function (Section 3.2) and the computation of error measures (Section 3.3).

### 3.1 Segmentation Algorithm

Time series segmentation algorithms partition a time series into several disjoint segments. Existing state-of-the-art time series segmentation algorithms can be classified into two categories: (i) Fix-length segmentation (FL) algorithms partition a time series based on fixed time windows. Thus, the segments produced by the FL have equal lengths. (ii) Variable-length segmentation algorithms, which are classified into three groups: the Top-down methods [38, 47], the Bottom-up ones [31, 32] and the Sliding-window ones [34, 28, 39, 22]. Among them, the Sliding-window (SW) has been proven to be more efficient than the Top-down and the Bottom-up methods [28, 29]. Thus, we choose the Sliding-window (SW) as the variable length segmentation algorithm. Figure 1 is produced by the SW method, which produces variable-length segments.

Note, there is no one-size-fits-all function family that can best model all kinds of time series data. For example, polynomials and ARMA models are better at modeling data from physical processes such as temperature [42, 12], while Gaussian functions are better for modeling relatively randomized data [33] such as stock prices. How to choose the best function family has been widely studied in prior work [48, 58, 15, 35] and will continue so. Nevertheless, as long as the compression techniques produce the common

**Table 1: Example TSA's for common statistics. Let $\boldsymbol{T}_1 = (a_1, b_1, [\cdots])$ and $\boldsymbol{T}_2 = (a_2, b_2, [\cdots])$ be the input time series in the time series analytic.**

| TSA Expression | Definition | Equivalent TSA Expression |
|---|---|---|
| Average $\mu_{T_1}$ '$\mu(T_1)$' | $\frac{1}{b_1 - a_1 + 1}(\sum_{i=a_1}^{b_1} T_1[i])$ | $\frac{1}{b_1 - a_1 + 1}(\text{Sum}(\boldsymbol{T_1}))$ |
| Standard Deviation $\sigma_{T_1}$ '$\sigma(T_1)$' | $\sqrt{\frac{1}{b_1 - a_1 + 1}\Big(\sum_{i=a_1}^{b_1}(T_1[i] - \mu_{T_1})^2\Big)}$ | $\sqrt{\frac{1}{b_1 - a_1 + 1} \times Sum(\boldsymbol{T_1} - \text{Constant}(\mu_{T_1})) \times Sum(\boldsymbol{T_1} - \text{Constant}(\mu_{T_1}))}$ |
| Correlation $r_{(T_1,T_2)}$ '$\text{Corr}(T_1,T_2)$' | $\frac{\sum_{i=max(a_1,a_2)}^{min(b_1,b_2)}\Big((T_1[i] - \mu_{T_1})(T_2[i] - \mu_{T_2})\Big)}{\sigma_{T_1} \times \sigma_{T_2}}$ | $\frac{Sum\Big((\boldsymbol{T_1} - \text{Constant}(\mu_{T_1})) \times (\boldsymbol{T_2} - \text{Constant}(\mu_{T_2}))\Big)}{\sigma_{T_1} \times \sigma_{T_2}}$ |
| Cross-correlation $r_{(T_1,T_2,m)}$ '$\text{CCorr}(T_1,T_2,m)$' | $\frac{\sum_{i=max(a_1,a_2+m)}^{min(b_1,b_2+m)}\Big((T_1[i] - \mu_{T_1})(T_2[i+m] - \mu_{T_2})\Big)}{\sigma_{T_1} \times \sigma_{T_2}}$ | $\frac{Sum\Big((\boldsymbol{T_1} - \text{Constant}(\mu_{T_1})) \times (Shift(\boldsymbol{T_2}, m) - \text{Constant}(\mu_{T_2}))\Big)}{\sigma_{T_1} \times \sigma_{T_2}}$ |
| Auto-correlation $r_{(T_1,m)}$ '$\text{ACorr}(T_1,m)$' | $\frac{\sum_{i=a_1+m}^{b_1}\Big((T_1[i] - \mu_{T_1})(T_1[i+m] - \mu_{T_1})\Big)}{\sigma^2_{T_1}}$ | $\frac{Sum\Big((\boldsymbol{T_1} - \text{Constant}(\mu_{T_1})) \times (Shift(\boldsymbol{T_1}, m) - \text{Constant}(\mu_{T_1}))\Big)}{\sigma_{T_1} \times \sigma_{T_1}}$ |

**Table 2: Grammar of time series analytic (TSA).**

| | |
|---|---|
| **Time Series Analytic (TSA)** | |
| Q $\rightarrow$ Ar | |
| \| Agg | |
| **Arithmetic Expression (Ar)** | |
| Ar $\rightarrow$ literal value in $R$ | |
| \| Ar $\otimes$ Ar | where $\otimes \in \{+, -, \times, \div\}$ |
| \| $\oslash$ Ar | where $\oslash \in \{\sqrt{}\}$ |
| \| Agg | |
| **Aggregation Expression (Agg)** | |
| Agg $\rightarrow$ Sum($\boldsymbol{T}, a', b'$) | $= \sum_{i=a'}^{b'} \boldsymbol{T}[i]$, when $\boldsymbol{T} = (a, b, [\ldots])$ and $[a', b'] \subseteq [a, b]$ |
| **Time Series Expression (TSE)** | |
| $\boldsymbol{T}$ $\rightarrow$ input time series | |
| \| Constant($v, a, b$) | $= (a, b, [\underbrace{v, \ldots, v}_{b-a+1}])$, $v$ is Ar expression |
| \| Shift($\boldsymbol{T}, k$) | $= (a + k, b + k, [\boldsymbol{T}[a], ..., \boldsymbol{T}[b]])$ |
| \| $\boldsymbol{T_1} + \boldsymbol{T_2}$ | $= (a, b, [\boldsymbol{T_1}[a] + \boldsymbol{T_2}[a], ..., \boldsymbol{T_1}[b] + \boldsymbol{T_2}[b]])$ where $\boldsymbol{T_1} = (a_1, b_1, [\ldots])$, $\boldsymbol{T_2} = (a_2, b_2, [\ldots])$, $a = \max(a_1, a_2)$ and $b = \min(b_1, b_2)$ |
| \| $\boldsymbol{T_1} - \boldsymbol{T_2}$ | $= (a, b, [\boldsymbol{T_1}[a] - \boldsymbol{T_2}[a], ..., \boldsymbol{T_1}[b] - \boldsymbol{T_2}[b]])$ |
| \| $\boldsymbol{T_1} \times \boldsymbol{T_2}$ | $= (a, b, [\boldsymbol{T_1}[a] \times \boldsymbol{T_2}[a], ..., \boldsymbol{T_1}[b] \times \boldsymbol{T_2}[b]])$ |

error metrics required by Plato, their compressions can be utilized by Plato and tight deterministic error guarantees will be provided.

## 3.2 Estimation Function Selection and Storage

Choosing an estimation function for a time series segment has two steps: (i) user identifies the function family, and (ii) Plato selects the best function in the family, i.e., the function that minimizes the Euclidean distance between the original values and the estimated values produced by the function. For example, if the user had chosen the 1st-degree polynomials as the compression function family, Plato will choose a polynomial $a_s \times i + b_s$ for each segment $s$. Consequently, Plato will store the coefficients $a_s$ and $b_s$ instead of storing all the values of the segments.

**Step 1: Function family selection.** The user chooses a function family $\mathbb{F}$ that is used for compressing a time series. In the present work, Plato uses the same function family for all the segments of a time series. Thus, the time series physical representation stores a token $\tau$ that identifies the function family.

**Step 2: Estimation function selection.** Any function $f$ in the chosen function family $\mathbb{F}$ is a *candidate estimation function*. Following the prior work [37, 5], Plato selects the candidate estimation function that minimizes the Euclidean distance between the original values and the estimated values produced by the function to be the final estimation function. More precisely,

$$f_T^* = \arg\min_{f \in \mathbb{F}} \Big(\sum_{i=a}^{b} (\boldsymbol{T}[i] - f(i))^2\Big)^{1/2} \quad (1)$$

EXAMPLE 3. *Given a time series* $\boldsymbol{T} = (1, 5, [0.2, 0.4, 0.4, 0.5, 0.6])$, *assume the function family identifier is "$p_1$" (i.e., "first-degree polynomial function family"). The functions* $f_1 = 0.05 \times i + 0.3$ *and* $f_2 = 0.09 \times i + 0.15$ *are two candidate estimation functions. Finally, Plato selects* $f_2 = 0.09 \times i + 0.15$ *as the estimation function since it produces the minimal Euclidean error, i.e.,* 0.0837.

**Function Representation (Physical) vs. Function (Logical).** Once an estimation function $f_T^*$ is selected, Plato stores the corresponding *function representation* $\tilde{f}_T^*$, which includes (i) the coefficients of the function $f_T^*$, and (ii) the function family identifier $\tau$. [2] For example, the function representation of the estimation function in Example 3 is $\tilde{f}_T^*$ = $((0.09, 0.15)$ , $p_1)$ where $p_1$ is a function family identifier indicating that the function family is "1-degree polynomial function family".

When we talk about the function itself logically, it can be regarded as a vector that maps time series: given a domain $[a, b]$, the vector $[f(a), f(a+1), \ldots, f(b)]$ maps a value to each position in the domain $[a, b]$. For example, consider

[2] All the segments in the same time series share one token $\tau$.

**Table 3: Error measures stored for a time series segment $T$ running from $a$ to $b$ and approximated with the estimation function $f_T^*$.**

| Error measures | Comments |
|---|---|
| $\|\varepsilon_T\|_2 = \sqrt{\sum_{i=a}^{b}(T[i] - f_T^*(i))^2}$ | $L_2$-norm of the estimation errors |
| $\|f_T\|_2 = \sqrt{\sum_{i=a}^{b}(f_T^*(i))^2}$ | $L_2$-norm of the estimated values |
| $\gamma_T = \|\sum_{i=a}^{b} T[i] - \sum_{i=a}^{b} f_T^*(i)\|$ | Absolute reconstruction error |

the estimation function $f_T^* = 0.09 \times i + 0.15$ in Example 3. Then $\boldsymbol{T} - f_T^* = [0.2 - f_T^*(1), 0.4 - f_T^*(2), 0.4 - f_T^*(3), 0.5 - f_T^*(4), 0.6 - f_T^*(5)] = [0.2 - 0.24, 0.4 - 0.33, 0.4 - 0.42, 0.5 - 0.51, 0.6 - 0.6] = [0.04, 0.07, -0.02, -0.01, 0]$.

## 3.3 Error Measures

In addition to the estimation function, Plato stores extra *error measures* $\Phi(T) = \{\|\varepsilon_T\|_2, \|f_T\|_2, \gamma_T\}$ for each time series segment $T$ (defined in domain $[a, b]$) where $\|\varepsilon_T\|_2$, $\|f_T\|_2$, and $\gamma_T$ are defined in Table 3.

EXAMPLE 4. *Consider the time series* $\boldsymbol{T} = (1, 5, [0.2, 0.4, 0.4, 0.5, 0.6])$ *in Example 3 again.* $f_T^* = 0.09 \times i + 0.15$ *is the estimation function. Thus* $\|\varepsilon_T\|_2 = \sqrt{\sum_{i=1}^{5}(T[i] - f_T^*(i))^2} = 0.0837$, $\|f_T\|_2 = \sqrt{\sum_{i=1}^{5}(f_T^*(i))^2} = 0.9813$, *and* $\gamma_T = |\sum_{i=1}^{5} T[i] - \sum_{i=1}^{5} f_T^*(i)| = 2.1 - 2.1 = 0$.

# 4. ONLINE ERROR GUARANTEE COMPUTATION

**Error Guarantee Definition.** Given a TSA $q$ involving time series $T_1, .., T_n$, let $R$ be the accurate answer of $q$ by executing $q$ directly on the original data points of $T_1, .., T_n$. Let $\hat{R}$ be the approximate answer of $q$ by executing $q$ on the compressed time series representations. Then $\varepsilon = |\hat{R} - R|$ is the *true error* of $q$. Notice that $\varepsilon$ is unknown since $R$ is unknown. An upper bound $\hat{\varepsilon}$ ($\hat{\varepsilon} \geq \varepsilon$) of the true error is called a *deterministic error guarantee* of $q$. With the help of $\hat{\varepsilon}$, we know that the accurate answer $R$ is within the range $[\hat{R} - \hat{\varepsilon}, \hat{R} + \hat{\varepsilon}]$ with 100% confidence. Plato provides *tight* deterministic error guarantees for each operator defined in Table 2 (Section 2).

**Error Guarantee Decomposition.** Recall that the TSA expression $q$ defined in Table 2 (Section 2) combines one or more time series aggregation operations via arithmetic operators, i.e., $q = Agg_1 \otimes Agg_2 \otimes \cdots \otimes Agg_n$ where $\otimes \in \{+, -, \times, \div, \sqrt{}\}$. In order to provide the deterministic error guarantee $\hat{\varepsilon}$ of the time series analytic $q$, the key step is to calculate the deterministic error guarantee $\hat{\varepsilon}_{Agg_i}$ of each aggregation operation $Agg_i$. Once we have $\hat{\varepsilon}_{Agg_i}$ for each aggregate expression, it is not hard to combine them to get the final error guarantee. Let $\hat{R}_{Agg_i}$ be the approximate answer of $Agg_i$ and $\hat{\varepsilon}_{Agg_i}$ be the corresponding error guarantee, Figure 4 summarizes the computation of the error guarantee for each arithmetic operator.

Given a TSA $Agg = \mathsf{Sum}(\boldsymbol{T})$ and the compressed time series representation $L_T = (\tilde{T^1}, ..., \tilde{T^k})$, when calculating $\hat{\varepsilon}_{Agg}$, there are two cases depending on whether $\boldsymbol{T}$ is an input time series or not.

| Operator | error guarantee |
|---|---|
| $Agg_1 + Agg_2$ | $\hat{\varepsilon}_{Agg_1} + \hat{\varepsilon}_{Agg_2}$ |
| $Agg_1 - Agg_2$ | $\hat{\varepsilon}_{Agg_1} + \hat{\varepsilon}_{Agg_2}$ |
| $Agg_1 \times Agg_2$ | $\hat{\varepsilon}_{Agg_1}\hat{R}_{Agg_2} + \hat{\varepsilon}_{Agg_2}\hat{R}_{Agg_1} + \hat{\varepsilon}_{Agg_1}\hat{\varepsilon}_{Agg_2}$ |
| $Agg_1 \div Agg_2$ | $\frac{\hat{\varepsilon}_{Agg_1}\hat{R}_{Agg_2} + \hat{\varepsilon}_{Agg_2}\hat{R}_{Agg_1}}{(\hat{R}_{Agg_2} - \hat{\varepsilon}_{Agg_2})\hat{R}_{Agg_2}}$ |
| $\sqrt{Agg_1}$ | $\sqrt{\hat{R}_{Agg_1} + \hat{\varepsilon}_{Agg_1}} - \sqrt{\hat{R}_{Agg_1}}$ |

**Figure 4: Error guarantee computation for arithmetic operators.**

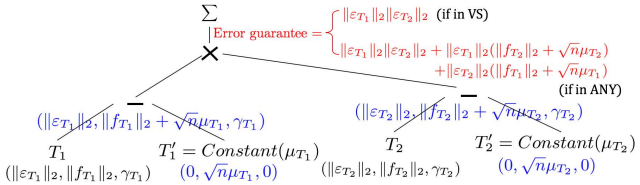



**Figure 5: Example of error measures propagation. Error measures in black color are precomputed offline during insertion time, while error measures in blue color are computed during the TSA processing time. The final error guarantees are in red color.**

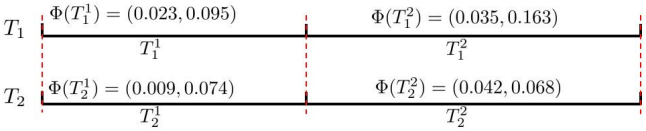



**Figure 6: Example of aligned segments.**

- Case 1. $\boldsymbol{T}$ is an input time series, then $\hat{\varepsilon}_{Agg} = \sum_{i=1}^{k} \gamma_{T^i}$ where $\gamma_{T^i}$ is the reconstruction error in the error measures of $T^i$. [3]
- Case 2. $\boldsymbol{T}$ is a derived time series by applying (recursively) the time series operators, i.e., $\mathsf{Constant}(v, a, b)$, $\mathsf{Shift}(\boldsymbol{T}, k)$, $\boldsymbol{T_1} + \boldsymbol{T_2}$, $\boldsymbol{T_1} - \boldsymbol{T_2}$ and $\boldsymbol{T_1} \times \boldsymbol{T_2}$. In this case, the aggregation expression $Agg = Sum(\boldsymbol{T})$ can be denoted as an expression tree. Figure 5 shows an example tree of the aggregation operator in the correlation TSA. In order to compute $\hat{\varepsilon}_{Agg}$, we first add the (precomputed) error measures $\Phi(\boldsymbol{T}) = (\|\varepsilon_T\|_2, \|f_T\|_2, \gamma_T)$ for the input time series in the tree and then propagate the error measures from the bottom time series to the root, using the tight error bound formulas explained in this section. Then we return the $\gamma_T$ in the $\Phi(\boldsymbol{T})$ as the final error guarantee.

For the general scenario where multiple segments are involved in each input time series[4] in the expression, there are two cases depending on whether the segments are aligned or not: If the $i$-th segment in $T_1$ has the same domain as the $i$-th segment in $T_2$ for all $i$, then $T_1$ and $T_2$ are *aligned*, otherwise, they are *misaligned*.

[3]Here we assume the aggregation operator aggregates the whole time series.

[4]Input time series means base time series.

## 4.1 Warmup: Error Guarantees for $T_1 + T_2$, $T_1 - T_2$, Shift and Constant

We first present the (easy to derive) error guarantees for $\mathsf{Constant}(v, a, b)$, $Shift(\boldsymbol{T}, k)$, $\boldsymbol{T_1 + T_2}$ and $\boldsymbol{T_1 - T_2}$.

For the time series $T$=$\mathsf{Constant}(v, a, b)$, the estimation function is $f^*_T = v$ [5]. Then the error measures stored by Plato are ($\|\varepsilon_T\|_2 = 0$, $\|f_T\|_2 = v\sqrt{b-a+1}$, $\gamma_T = 0$). Note, the error guarantee for $\hat{\varepsilon}_{Sum(\mathsf{Constant}(v,a,b))}$ is 0, since $\gamma_T = 0$.

For the time series $T$=$\mathsf{Shift}(T, k)$, we simply propagate the error measures ($\|\varepsilon_T\|_2$, $\|f_T\|_2$, $\gamma_T$) of the input $T$.

Given time series $T_1 = (T_1^1, ..., T_1^{k_1})$ and $T_2 = (T_2^1, ..., T_2^{k_2})$, let ($\|\varepsilon_{T_1^i}\|_2, \|f_{T_1^i}\|_2$, $\gamma_{T_1^i}$) and ($\|\varepsilon_{T_2^j}\|_2, \|f_{T_2^j}\|_2, \gamma_{T_2^j}$) be the computed error measures for the segments $T_1^i$ and $T_2^j$ respectively. The error measures for $\boldsymbol{T_1 + T_2}$ (and identical for $\boldsymbol{T_1 - T_2}$) are ($\sum_i^{k_1} \|\varepsilon_{T_1^i}\|_2 + \sum_i^{k_2} \|\varepsilon_{T_2^i}\|_2$, $\sum_i^{k_1} \|f_{T_1^i}\|_2 + \sum_i^{k_2} \|f_{T_2^i}\|_2$, $\sum_i^{k_1} \gamma_{T_1^i} + \sum_i^{k_2} \gamma_{T_2^i}$). Therefore, the error guarantee $\hat{\varepsilon}_{Sum(\boldsymbol{T_1+T_2})} = \hat{\varepsilon}_{Sum(\boldsymbol{T_1-T_2})} = \sum_i^{k_1} \gamma_{T_1^i} + \sum_i^{k_2} \gamma_{T_2^i}$ regardless of whether $T_1$ and $T_2$ are aligned or misaligned.

## 4.2 Error Guarantee of $\hat{\varepsilon}_{Sum(T_1 \times T_2)}$

Next, we show how to compute the challenging error guarantee $\hat{\varepsilon}_{Sum(T_1 \times T_2)}$ in both aligned and misaligned cases in Section 4.3 and Section 4.4 respectively.

In the following derivation and proof of the error guarantees, we use the following definitions and intuitions.

First, the time series $\boldsymbol{T} = (a, b, [\boldsymbol{T}[a], ..., \boldsymbol{T}[b]])$, the estimation function $f^*_T$ of $T$ which provides the vector of estimated values $[f^*_T[a], \ldots, f^*_T[b]]$, and the vector of errors $\boldsymbol{\varepsilon}_T = \boldsymbol{T} - f^*_T = (a, b, [\boldsymbol{T}[a] - f^*_T(a), ..., \boldsymbol{T}[b] - f^*_T(b)])$ produced by the estimation function are all considered vectors of $b - a + 1$ dimensions. Note that the estimation function is treated, for the purpose of proving the error guarantees formulas, as the vector of predicted values and not as the vector of coefficients. For example, a 1-st degree polynomial function that estimates values in the range $10 - 20$ should be thought of as an 11-dimension vector and not as a 2-dimension vector. Nevertheless, this 11-dimension vector is only conceptually used in the derivation and proof of the error guarantees formulas. It is never produced in the actual computations.

Second, given two vectors $f_1$ and $f_2$, we denote by $\langle f_1, f_2 \rangle = \sum_{i=a}^{b} f_1(i) f_2(i)$ the inner product of $f_1$ and $f_2$.

Third, the *restriction* operator allows us to think of logical segments, regardless of whether there are respective physical segments. This will become useful in the derivation of formulas for misaligned segments. Formally, let $V|_{[a,b]}$ be the *restriction* operation, which restricts a vector $V$ to the domain $[a, b]$. Recall a time series segment is a subsequence of a time series, for which we actually store data. Thus, a segment is also logically the *restriction* of a time series $\boldsymbol{T}$ from a bigger domain $[a, b]$ into a smaller domain $[a', b'] \subseteq [a, b]$, denoted as $\boldsymbol{T}|_{[a',b']}$. Figure 3(b) visualizes the restriction operator. For example, consider a time series $\boldsymbol{T} = (1, 4, [1.2, 1.3, 1.3, 1.2])$, then $\boldsymbol{T}|_{[2,3]} = (2, 3, [1.3, 1.3])$ is a restriction of $\boldsymbol{T}$. It is $\boldsymbol{T}|_{[a',b']}[i] = T[i]$ for all $i \in [a', b']$.

## 4.3 Error Guarantee on Aligned Segments

Given two aligned time series $T_1 = (T_1^1, ..., T_1^k)$ and $T_2 = (T_2^1, ..., T_2^k)$ where $T_1^i = T_1|_{[a_i,b_i]}$ and $T_2^i = T_2|_{[a_i,b_i]}$, let ($\|\varepsilon_{T_1^i}\|_2, \|f_{T_1^i}\|_2, \gamma_{T_1^i}$) and ($\|\varepsilon_{T_2^i}\|_2, \|f_{T_2^i}\|_2, \gamma_{T_2^i}$) be the computed error measures for segment $T_1^i$ and $T_2^i$ respectively, then the error guarantee of $\mathsf{Sum}(\boldsymbol{T_1 \times T_2})$ for any estimation function family, is:

$$\begin{aligned}
\varepsilon &= \Big| \sum_{i=a}^{b} \boldsymbol{T}_1[i]\boldsymbol{T}_2[i] - \sum_{i=a}^{b} f^*_{T_1}(i) f^*_{T_2}(i) \Big| \\
&= \Big| \sum_{i=1}^{k} \Big( \sum_{j=a_i}^{b_i} \boldsymbol{T}_1[i]\boldsymbol{T}_2[i] - \sum_{j=a_i}^{b_i} f^*_{T_1}(i) f^*_{T_2}(i) \Big) \Big| \\
&= \Big| \sum_{i=1}^{k} \Big( \langle \boldsymbol{\varepsilon}_{T_1^i}, f^*_{T_2^i} \rangle + \langle \boldsymbol{\varepsilon}_{T_2^i}, f^*_{T_1^i} \rangle + \langle \boldsymbol{\varepsilon}_{T_1^i}, \boldsymbol{\varepsilon}_{T_2^i} \rangle \Big) \Big| \\
&\leq \sum_{i=1}^{k} \Big( \|\varepsilon_{T_1^i}\|_2 \|\varepsilon_{T_2^i}\|_2 + \|\varepsilon_{T_1^i}\|_2 \|f_{T_2^i}\|_2 + \|f_{T_1^i}\|_2 \|\varepsilon_{T_2^i}\|_2 \Big)
\end{aligned} \tag{2}$$

The last inequality is obtained by Applying the Hölder inequality [11].

Example 5. *Consider the two aligned time series in Figure 6. Both $T_1$ and $T_2$ are partitioned into two segments in this case, i.e., $(T_1^1, T_1^2)$ and $(T_2^1, T_2^2)$. Plato stores the error measures $\Phi(T_i^j)$ for each segment $T_i^j$. For instance, $\Phi(T_1^1) = (\|\varepsilon_{T_1^1}\|_2, \|f_{T_1^1}\|_2, \gamma_{T_1^1}) = (0.023, 0.95, 0)$. Then the error guarantee of $\mathsf{Sum}(\boldsymbol{T_1 \times T_2})$ on $T_1$ and $T_2$ is computed as $(\|\varepsilon_{T_1^1}\|_2\|\varepsilon_{T_2^1}\|_2 + \|\varepsilon_{T_1^1}\|_2\|f_{T_2^1}\|_2 + \|f_{T_1^1}\|_2\|\varepsilon_{T_2^1}\|_2) + (\|\varepsilon_{T_1^2}\|_2\|\varepsilon_{T_2^2}\|_2 + \|\varepsilon_{T_1^2}\|_2\|f_{T_2^2}\|_2 + \|f_{T_1^2}\|_2\|\varepsilon_{T_2^2}\|_2) = (0.023 \times 0.009 + 0.023 \times 0.074 + 0.095 \times 0.009) + (0.035 \times 0.042 + 0.035 \times 0.068 + 0.163 \times 0.042) = 0.01346$.*

### 4.3.1 Orthogonal projection optimization

If the estimation function family forms a vector space ($\mathsf{VS}$), [6] then we can apply the *orthogonal projection property* in VS to significantly reduce the error guarantee of $sum(T_1 \times T_2)$ from Formula 2 to Formula 3.

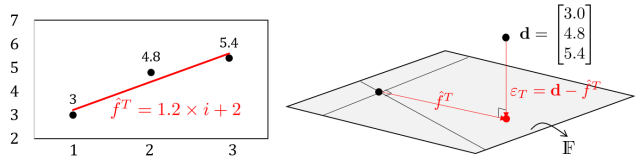
$$\begin{aligned}
\varepsilon &= \Big| \sum_{i=1}^{k} \Big( \underbrace{\langle \boldsymbol{\varepsilon}_{T_1^i}, f^*_{T_2^i} \rangle}_{=0 \ in \ VS} + \underbrace{\langle \boldsymbol{\varepsilon}_{T_2^i}, f^*_{T_1^i} \rangle}_{=0 \ in \ VS} + \langle \boldsymbol{\varepsilon}_{T_1^i}, \boldsymbol{\varepsilon}_{T_2^i} \rangle \Big) \Big| \\
&\leq \sum_{i=1}^{k} \Big( \|\varepsilon_{T_1^i}\|_2 \|\varepsilon_{T_2^i}\|_2 \Big)
\end{aligned} \tag{3}$$

Example 6. *Consider the two aligned time series in Figure 6 again. The estimation function family is polynomial function family, which is in $\mathsf{VS}$. Based on Formula 3, the error guarantee for $\mathsf{Sum}(\boldsymbol{T_1 \times T_2})$ is $\|\varepsilon_{T_1^1}\|_2 \times \|\varepsilon_{T_2^1}\|_2 + \|\varepsilon_{T_1^2}\|_2 \times \|\varepsilon_{T_2^2}\|_2 = 0.023 \times 0.009 + 0.035 \times 0.042 = 0.001677$. This error guarantee is about $8\times$ smaller than that in Example 5 (i.e., 0.01346), where we did not take into account that the function family is VS.*

Example 6 indicates the power of the orthogonal projection optimization. Lemma 1 is the key step to proving Formula 3.

[5] Under the reasonable assumption that any practical family will also include the constant function.

[6] A vector space is a set that is closed under finite vector addition and scalar multiplication. `http://mathworld.wolfram.com/VectorSpace.html`.




(a) Example estimation function (b) Visualization of orthogonal property

**Figure 7: (a) shows the estimation function for three data points. (b) visualizes the orthogonal projection of the three data points onto the 2-dimensional plane $\mathbb{F}$.**

LEMMA 1. *(Orthogonal Projection Property) Let $\mathbb{F}$ be a function family that forms a vector space VS and $f_T^* \in \mathbb{F}$ be the estimation function of time series $T$. Then $f_T^*$ is the orthogonal projection of $\mathbf{T}$ onto $\mathbb{F}$ [44].*

Lemma 1 implies that $\boldsymbol{\varepsilon}_T$ is orthogonal to any function $f_T \in \mathbb{F}$, which means $\langle \boldsymbol{\varepsilon_T}, \boldsymbol{f_T} \rangle = 0$. Therefore, given any two aligned segments $T_1^i$ and $T_2^i$, as both $f_{T_1^i}^*$ and $f_{T_2^i}^*$ are in VS, it follows that $\langle \boldsymbol{\varepsilon}_{T_1^i}, f_{T_2^i}^* \rangle = 0$ and $\langle \boldsymbol{\varepsilon}_{T_2^i}, f_{T_1^i}^* \rangle = 0$. Thus we managed to remove two of the three terms of the first line in Equation 2. The benefit we got was much more than 3 times - it was about 8 times. The reason is that the remaining formula in Equation 3 is *Amplitude-Independent (AI)*. An error guarantee is called amplitude-independent (AI) if it does *not* directly use $\|f_T\|_2$ or, more generally, does not use a quantity that grows with the size of the $f_T$ as opposed to growing with the size of the error. Since the value of $\|f_T\|_2$ reflects the absolute values of the data points in $T$, it is normal to be much larger than the $\|\varepsilon_T\|_2$. Thus, non-AI error guarantees usually are much more loose than AI error guarantees. In summary, the importance of VS is that it allows the orthogonal projection optimization.

Given the importance of the orthogonal property, we visualize the intuition behind it. Consider a time series with three data points $T = (1, 3, [3.0, 4.8, 5.4])$ and let $\mathbb{F}$ be the 1-degree polynomial function family (i.e., 2-dimensional). The estimation function that minimizes the error to the original data is $f_T^* = 1.2 \times i + 2$ (Figure 7(a)). As shown in Figure 7(b), $f_T^*$ is the orthogonal projection of $\boldsymbol{T}$ onto $\mathbb{F}$. The error vector is $\boldsymbol{\varepsilon}_T = (-0.2, 0.4, -0.2)$. Based on Lemma 1, for any candidate estimation function $f = \alpha \times i + \beta$ $(\alpha, \beta \in R)$, we have $\langle \boldsymbol{\varepsilon_T}, \boldsymbol{f} \rangle = 0.8\alpha - 0.8\alpha + 0.4\beta - 0.4\beta = 0$.

**Elimination of $\gamma_T$.** We can get an extra benefit from the orthogonal projection property: saving space. The error measure $\gamma_T$ can be avoided as it is guaranteed to be 0. This is because $\gamma_T = \langle T - f_T^*, 1 \rangle$ and 1 is a constant function in the function family in VS. According to Lemma 1, we know $\langle T - f_T^*, 1 \rangle = 0$. Therefore, we have $\gamma_T = 0$.

## 4.4 Error Guarantee on Misaligned Segments

The time series involved in TSA expressions are usually misaligned due to the following two reasons: First, variable length segmentation leads to better compression but variable length segments of different time series are generally misaligned. Thus operations, like corrrelation and cross-correlation that use multiple input time series will need to produce good error guarantees despite the misalignment. Second, the cross-correlation and the auto-correlation require time shifting the time series. Even when the original time series segmentations are aligned, they will be misaligned after time shifting. Given the big application of cross-correlation in mining cases where a first process is causing (with a time shift) a second process, it is paramount that the Plato techniques also provide guarantees for misaligned segments.

$$
\begin{aligned}
\hat{\varepsilon} &= \Big| \sum_{i=a}^{b} \boldsymbol{T}_1[i]\boldsymbol{T}_2[i] - \sum_{i=a}^{b} f_{T_1}^*(i) f_{T_2}^*(i) \Big| \\
&\le \Big| \langle \boldsymbol{\varepsilon_{T_1}}, f_{T_2}^* \rangle \Big| + \Big| \langle \boldsymbol{\varepsilon_{T_2}}, f_{T_1}^* \rangle \Big| + \Big| \langle \boldsymbol{\varepsilon_{T_1}}, \boldsymbol{\varepsilon_{T_2}} \rangle \Big| \\
&= \Big| \sum_{i=1}^{k_1} \langle \boldsymbol{\varepsilon_{T_1^i}}, f_{T_2}^*|_{[a_1^i, b_1^i]} \rangle \Big| + \Big| \sum_{i=1}^{k_2} \langle \boldsymbol{\varepsilon_{T_2^i}}, f_{T_1}^*|_{[a_2^i, b_2^i]} \rangle \Big| + \Big| \langle \boldsymbol{\varepsilon_{T_1}}, \boldsymbol{\varepsilon_{T_2}} \rangle \Big| \\
&\le \sum_{i=1}^{k_1} \Big( \|\varepsilon_{T_1^i}\|_2 \big( \sum_{j \in \Pi_{T_2, [a_1^i, b_1^i]}} \|f_{T_2^j}\|_2^2 \big)^{\frac{1}{2}} \Big) + \sum_{i=1}^{k_2} \Big( \|\varepsilon_{T_2^i}\|_2 \big( \sum_{j \in \Pi_{T_1, [a_2^i, b_2^i]}} \|f_{T_1^j}\|_2^2 \big)^{\frac{1}{2}} \Big) \quad (1) \\
&\quad + \Big| \langle \boldsymbol{\varepsilon_{T_1}}, \boldsymbol{\varepsilon_{T_2}} \rangle \Big| \quad (2) \qquad (4)
\end{aligned}
$$

Let $\Pi_{T,[a,b]}$ be the set of segments in $T$ covering the domain $[a, b]$. For example, consider the two misaligned time series $T_1$ and $T_2$ and the segments $T_2^1$ and $T_2^2$ in $T_2$ cover the domain $[a_1^1, b_1^1]$, then $\Pi_{T_2,[a_1^1,b_1^1]} = \{T_2^1, T_2^2\}$ . If any kinds of compression function families are allowed, i.e., the functions belong in ANY, the error guarantee $\hat{\varepsilon}$ of $\mathsf{Sum}(\boldsymbol{T_1} \times \boldsymbol{T_2})$ on misaligned time series is shown in Formula 4. Formula 4 is a stepping stone towards producing the final formula as the computation of $|\langle \boldsymbol{\varepsilon_{T_1}}, \boldsymbol{\varepsilon_{T_2}} \rangle|$ (Formula 4②) has not been given yet. It will be discussed in Section 4.4.1. Section 4.4.2 discusses how to apply the orthogonal property optimization to improve Formula 4①.

### 4.4.1 Segment combination selection

To compute $|\langle \boldsymbol{\varepsilon_{T_1}}, \boldsymbol{\varepsilon_{T_2}} \rangle|$, a straightforward method (called IS) is to use the domains of segments in $T_1$ and $T_2$ independently, then choose the one with minimal value. Let's first see how to compute $|\langle \boldsymbol{\varepsilon_{T_1}}, \boldsymbol{\varepsilon_{T_2}} \rangle|$ with the domains of segments in $T_1$.

$$
\begin{aligned}
|\langle \boldsymbol{\varepsilon_{T_1}}, \boldsymbol{\varepsilon_{T_2}} \rangle| &\le \sum_{i=1}^{k_1} \Big| \langle \boldsymbol{\varepsilon_{T_1}}|_{[\boldsymbol{a_1^i}, \boldsymbol{b_1^i}]}, \boldsymbol{\varepsilon_{T_2}}|_{[\boldsymbol{a_1^i}, \boldsymbol{b_1^i}]} \rangle \Big| \\
&= \sum_{i=1}^{k_1} \Big| \langle \boldsymbol{\varepsilon_{T_1^i}}, \boldsymbol{\varepsilon_{T_2}}|_{[\boldsymbol{a_1^i}, \boldsymbol{b_1^i}]} \rangle \Big| \\
&\le \sum_{i=1}^{k_1} \big( \|\varepsilon_{T_1^i}\|_2 \big( \sum_{j \in \Pi_{T_2, [a_1^i, b_1^i]}} \|\varepsilon_{T_2^j}\|_2^2 \big)^{\frac{1}{2}} \big)
\end{aligned}
$$

In the last step of the above Formula, $T_2|_{[a_1^i, b_1^i]}$ is not a segment that Plato precomputed in $T_2$. Thus, we need to use all the segments in $T_2$ covering $[a_1^i, b_1^i]$, i.e., $\Pi_{T_2,[a_1^i,b_1^i]}$. Similarly, we can compute $|\langle \boldsymbol{\varepsilon_{T_1}}, \boldsymbol{\varepsilon_{T_2}} \rangle|$ according to the domains of segments in $T_2$. Finally, IS chooses the minimal one between them. However, IS does not produce tight guarantees. The reason is that among all possible segmentations, it arbitrarily chooses to use the segmentation of $T_1$ as the driver - as it becomes apparent from the outer summation over $k_1$. Thus Plato does not use it. Next, we show the tight computation called *OS*, which is used by Plato.

**Optimal strategy (OS)** OS (Algorithm 1) first computes an error distribution array $E_{T_1}$ (resp. $E_{T_2}$) for $T_1$ (resp. $T_2$)

**Algorithm 1:** Optimal segment combination (OS)

**Input:** Compressed segment representations $L_{T_1}$, $L_{T_2}$
**Output:** A segment combination $OPT$

1 $\varepsilon_1 = 0$, $\varepsilon_2 = 0$, $i_i = 0$, $i_2 = 0$, $start = 0$, $OPT = \emptyset$, $current = \emptyset$;
2 Compute $E_{T_1}$ and $E_{T_2}$;
3 **while** $i_1 < k_1$ *or* $i_2 < k_2$ **do**
4  **if** $b_1^{i_1} \le b_2^{i_2}$ **then**
5  $\varepsilon_1 += E_{T_1}[i_1++]$;
6  **else**
7  $\varepsilon_2 += E_{T_2}[i_2++]$ ;
8  **if** $\varepsilon_1 \le \varepsilon_2$ *AND* $b_1^{i_1} \ge b_2^{i_2}$ **then**
9  $current = [start, b_1^{i_1}]$;
10  $OPT \leftarrow OPT \cup \{current\}$;
11  $start = b_1^{i_1} + 1$;
12  $\varepsilon_2 \leftarrow \varepsilon_1$;
13  **if** $\varepsilon_2 \le \varepsilon_1$ *AND* $b_2^{i_2} \ge b_1^{i_1}$ **then**
14  $current = [start, b_2^{i_2}]$;
15  $OPT \leftarrow OPT \cup \{current\}$;
16  $start = b_2^{i_2} + 1$;
17  $\varepsilon_1 \leftarrow \varepsilon_2$;
18 Return $OPT$;

(line 2) according to the domains of the segments as follows:

$$E_{T_1} = \left\{ \|\varepsilon_{T_1^i}\|_2 \times \Big( \sum_{j \in \Pi_{T_2,[a_1^i, b_1^i]}} \|\varepsilon_{T_2^i}\|_2^2 \Big)^{\frac{1}{2}} \Big| 1 \le i \le k_1 \right\}$$

$$E_{T_2} = \left\{ \|\varepsilon_{T_2^i}\|_2 \times \Big( \sum_{j \in \Pi_{T_1,[a_2^i, b_2^i]}} \|\varepsilon_{T_1^i}\|_2^2 \Big)^{\frac{1}{2}} \Big| 1 \le i \le k_2 \right\}$$

Then OS increases $\varepsilon_1$ (resp. $\varepsilon_2$) by adding the values from $E_{T_1}$ (resp. $E_{T_2}$) (lines 4-7) and checks whether the current domain achieves the minimal errors (lines 8-17). If yes, OS adds the current domain (either $[start, b_1^{i_1}]$ or $[start, b_2^{i_2}]$) to the final segment combination list. After that, OS starts from a new domain and repeats the previous steps until all the segments are processed. The time complexity of OS is $O(k_1 + k_2)$.

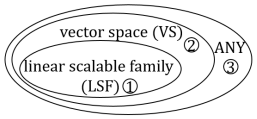
Let $OPT(L_{T_1}, L_{T_2})$ be the segment combination returned by OS. Then $|\langle \boldsymbol{\varepsilon_{T_1}}, \boldsymbol{\varepsilon_{T_2}} \rangle|$ is computed as follows:

$$\sum_{[a,b] \in OPT(L_{T_1}, L_{T_2})} \Big( \Big( \sum_{i \in \Pi_{T_1,[a,b]}} \|\boldsymbol{\varepsilon_{T_1^i}}\|_2^2 \Big)^{\frac{1}{2}} \Big( \sum_{i \in \Pi_{T_2,[a,b]}} \|\boldsymbol{\varepsilon_{T_2^i}}\|_2^2 \Big)^{\frac{1}{2}} \Big)$$

OS provides the optimal segment combination that produces the minimum $|\langle \boldsymbol{\varepsilon_{T_1}}, \boldsymbol{\varepsilon_{T_2}} \rangle|$.

### 4.4.2 Orthogonal projection optimization

In this part, we present how to apply orthogonal property optimization to improve Formula 4①. Recall that in the aligned case (if the function family is in VS) we can apply the orthogonal property optimization to guarantee $\langle \varepsilon_{T_1^i}, f^*_{T_2}|_{[a_1^i, b_1^i]} \rangle = 0$. This is because $f^*_{T_2}|_{[a_1^i, b_1^i]} = f^*_{T_2^i}$, which is a function in the family. However, in the misaligned case $\langle \varepsilon_{T_1^i}, f^*_{T_2}|_{[a_1^i, b_1^i]} \rangle$ cannot be guaranteed to be 0 since $f^*_{T_2}|_{[a_1^i, b_1^i]}$ may not be a function in the family. Indeed, the restriction of the estimation function $f^*_{T_2}$ to this sub-domain $f^*_{T_2}|_{[a_1^1, b_1^1]}$ may not be a function in the family anymore.




(a) Function family groups

| Family Group | Example Function Family |
|---|---|
| LSF① | $\{ax + b \mid a, b \in R\}$ |
| VS\LSF② | $\{a \ln(1 + x) + b \mid a, b \in R\}$ |
| ANY\VS③ | $\{\frac{L}{1+\exp(ax+b)} + c \mid L, a, b, c \in R\}$ $\{a \exp(\frac{-(x-b)^2}{2c^2}) + d \mid a, b, c, d \in R\}$ |

(b) Example function families

**Figure 8: Function family groups and examples.**

To guarantee the restriction of the function from a bigger domain to a smaller domain is still in the same function family, we identify a function family group called linear scalable function family (LSF), which is subset of VS but superset of the polynomial function family.

**Linear Scalable Function Family (LSF).** Informally, a linear scalable family is a function family such that for any function $f$ in that family and any shift $a - a'$, there is a function $f'$ in that family such that $f'(x + a - a') = f(x)$ for all $x$ in the domain. Definition 1 gives the formal definition.

Definition 1 (Linear scalable family (LSF)). *Let $\mathbb{F}$ be a function family defined in domain $[a, b]$. $\mathbb{F}$ is a linear scalable family if for any function $f \in \mathbb{F}$ and any range $[a', b'] \subseteq [a, b]$, there exists a function $f' \in \mathbb{F}$ such that $\mathsf{Shift}(f|_{[a',b']}, a - a') = f'|_{[a,a+b'-a']}$.*

Lemma 2. *The polynomial family belongs to the linear scalable family.*

Proof. Let $\mathbb{F} = \{\sum_i \alpha_i t^i \mid \alpha_i \in R\}$ be a polynomial function family defined on $[a, b]$. The restriction of $f \in \mathbb{F}$ on $[a', b'] \subseteq [a, b]$ is $f|_{[a',b']} = (a', b', [\sum_i \alpha_i (a')^i, ..., \sum_i \alpha_i (b')^i])$. The shift of $f|_{[a',b']}$ to $a - a'$ steps is $\mathsf{Shift}(f|_{[a',b']}, a - a') = (a, a + b' - a', [\sum_i \alpha_i (a')^i, ..., \sum_i \alpha_i (b')^i])$. $[\sum_i \alpha_i (a')^i, ..., \sum_i \alpha_i (b')^i]$ can be transformed into $[\sum_i \beta_i (a)^i, ..., \sum_i \beta_i (a + b' - a')^i]$ such that $\beta_i = \frac{\alpha_i (a'+k)^i}{(a+k)^k}$ for all $i \in [a, a + b' - a']$. Let $f' = \sum_i \beta_i t^i$ be a function in $\mathbb{F}$. Thus $f'|_{[a,a+b'-a']} = [\sum_i \beta_i (a)^i, ..., \sum_i \beta_i (a + b' - a')^i] = Shift(f|_{[a',b']}, a - a')$. □

In this paper, we study and distinguish three different function family groups, i.e., ANY, VS, and LSF. Figure 8 shows the relation of the three function family groups and also provides example function families for each group.

In the following, we present how to use the orthogonal projection optimization in the misaligned case to improve Formula 4①. Let $\boldsymbol{f_{T_1}}$ (resp. $\boldsymbol{f_{T_2}}$) be the function created from the concatenation of the individual estimation functions on the segments $T_1^i$ ($i \in [1, k_1]$) (resp. $T_2^j$ ($j \in [1, k_2]$)). That is $\boldsymbol{f_{T_1}}|_{[\boldsymbol{a_1^i}, \boldsymbol{b_1^i}]} = f^*_{T_1^i}$ for all $i \in [1, k_1]$ and $\boldsymbol{f_{T_2}}|_{[\boldsymbol{a_2^i}, \boldsymbol{b_2^i}]} = f^*_{T_2^i}$ for all $i \in [1, k_2]$. Then the Equation 4① in the misaligned environment can be reduced as follows. We highlight the parts that would disappear if the segments were aligned.

$$\sum_{i=1}^{k_1} \Big( \|\varepsilon_{T_1^i}\|_2 \times \overbrace{\|\boldsymbol{f_{T_2}}|_{[\boldsymbol{a_1^i}, \boldsymbol{b_1^i}]} - \boldsymbol{f^*_{T_1^i}}\|_2}^{=0 \ if \ aligned} \Big)$$
$$+ \sum_{i=1}^{k_2} \Big( \|\varepsilon_{T_2^i}\|_2 \times \overbrace{\|\boldsymbol{f_{T_1}}|_{[\boldsymbol{a_2^i}, \boldsymbol{b_2^i}]} - \boldsymbol{f^*_{T_2^i}}\|_2}^{=0 \ if \ aligned} \Big) \qquad (5)$$

The proof of the tightness is in Appendix G [6].

**Elimination of $\|f_T\|_2$.** If $T$ is compressed by a function in LSF [7], then $\|f_T\|_2$ can be safely eliminated. This is because

[7] And we know that it many only be combined with other segments compressed by a function in LSF.

**Table 4: Error guarantees for the time series analytic (TSA) $Sum(T_1 \diamond T_2)$ where $\diamond \in \{\times, +, -\}$ on both aligned and misaligned time series compressed by estimation functions in different families. We assume $T_1$ and $T_2$ have $k_1$ and $k_2$ segments respectively. In the aligned case, we have $k_1 = k_2 = k$. $OPT(L_{T_1}, L_{T_2})$ is the optimal segment combination returned by the algorithm OS in Section 4.4.1**

| | function family | error guarantees on aligned time series | AI | Tight | error guarantees on misaligned time series | AI | Tight |
|---|---|---|---|---|---|---|---|
| $Sum(\boldsymbol{T_1} \times \boldsymbol{T_2})$ | ANY\VS | $\sum_{i=1}^{k} \left(\|\varepsilon_{T_1^i}\|_2 \times \|\varepsilon_{T_2^i}\|_2\right) + \sum_{i=1}^{k} \left(\|\varepsilon_{T_1^i}\|_2 \times \|f_{T_2^i}\|_2\right) + \sum_{i=1}^{k} \left(\|\varepsilon_{T_2^i}\|_2 \times \|f_{T_1^i}\|_2\right)$ | ✗ | ✓ | $\sum_{i=1}^{k_1} \left(\|\varepsilon_{T_1^i}\|_2 \times (\sum_{j \in \Pi_{T_2,[a_1^i,b_1^i]}} \|f_{T_2^j}\|_2^2)^{\frac{1}{2}}\right) + \sum_{i=1}^{k_2} \left(\|\varepsilon_{T_2^i}\|_2 \times (\sum_{j \in \Pi_{T_1,[a_2^i,b_2^i]}} \|f_{T_1^j}\|_2^2)^{\frac{1}{2}}\right) + \sum_{[a,b] \in OPT(L_{T_1}, L_{T_2})} \left((\sum_{i \in \Pi_{T_1,[a,b]}} \|\varepsilon_{T_1^i}\|_2^2)^{\frac{1}{2}} \times (\sum_{i \in \Pi_{T_2,[a,b]}} \|\varepsilon_{T_2^i}\|_2^2)^{\frac{1}{2}}\right)$ | ✗ | ✓ |
| | VS\LSF | $\sum_{i=1}^{k} \left(\|\varepsilon_{T_1^i}\|_2 \times \|\varepsilon_{T_2^i}\|_2\right)$ | ✓ | ✓ | | | |
| | LSF | | | | $\sum_{i=1}^{k_1} \left(\|\varepsilon_{T_1^i}\|_2 \times \|\boldsymbol{f_{T_2}}|_{[\boldsymbol{a_1^i}, \boldsymbol{b_1^i}]} - \boldsymbol{f^*_{T_1^i}}\|_2\right) + \sum_{i=1}^{k_2} \left(\|\varepsilon_{T_2^i}\|_2 \times \|\boldsymbol{f_{T_1}}|_{[\boldsymbol{a_2^i}, \boldsymbol{b_2^i}]} - \boldsymbol{f^*_{T_2^i}}\|_2\right) + \sum_{[a,b] \in OPT(L_{T_1}, L_{T_2})} \left((\sum_{i \in \Pi_{T_1,[a,b]}} \|\varepsilon_{T_1^i}\|_2^2)^{\frac{1}{2}} \times (\sum_{i \in \Pi_{T_2,[a,b]}} \|\varepsilon_{T_2^i}\|_2^2)^{\frac{1}{2}}\right)$ | ✓ | ✓ |
| $Sum(\boldsymbol{T_1} + \boldsymbol{T_2})$ / $Sum(\boldsymbol{T_1} - \boldsymbol{T_2})$ | ANY | $\sum_{i=1}^{k} (\gamma_{T_1^i} + \gamma_{T_2^i})$ | ✓ | ✓ | $\sum_{i=1}^{k_1} \gamma_{T_1^i} + \sum_{j=1}^{k_2} \gamma_{T_2^j}$ | ✓ | ✓ |

**Table 5: Data Characteristics**

| | avg # of data points in each time series | # of time series | resolution |
|---|---|---|---|
| HF | 126,059,817 | 15 | millisecond |
| HI | 2,676,311 | 14 | second |
| HB | 1,669,835 | 16 | minute |
| HA | 1,587,258 | 11 | minute |

**Table 6: Number of coefficients and error measures**

| | # of coefficients | # of error measures |
|---|---|---|
| Polynomial | 2 | 1 |
| Gaussian | 4 | 3 |

**Table 7: Compression time (in milliseconds).**

| | $10^4$ points | $10^5$ points | $10^6$ points | $10^7$ points |
|---|---|---|---|---|
| FL | 0.78 | 7.09 | 61.74 | 586.33 |
| SW | 1.47 | 11.02 | 105.95 | 1009.25 |

the error guarantees provided by LSF can get rid of $\|f_T\|_2$ while those given by ANY or VS rely on $\|f_T\|_2$. Notice that, getting rid of $\|f_T\|_2$ makes the produced error guarantees to be Amplitude-independent (AI). Thus we highly suggest users to choose function families in LSF when compressing time series.

Table 4 summarizes the error guarantees for TSAs on both aligned and misaligned time series.

# 5. EXPERIMENTS

## 5.1 Environment and Setting

All experiments were conducted on a computer with a $4^{th}$ Intel $i7$-4770 processor (3.6 GHz), 16 GB RAM, running Ubuntu 14.04.1. The algorithms were implemented in $C++$ and were compiled with $g++$ 4.8.4.

**Dataset.** We evaluated all the error guarantee methods on four real-life datasets: Historical Forex Data (HF), Historical IoT Data (HI), Historical Bitcoin Exchanges Data (HB), and Historical Air Quality Data (HA). Table 5 summarizes the data characteristics [8]. The detailed description of each dataset is presented in Appendix I in the full version [6].

**Segmentation algorithms.** We adopt the fixed-length segmentation (FL) and the sliding window algorithm (SW). The segments produced by the FL have equal lengths, and will be utilized in our aligned experiments, while the segments created by the SW have variable lengths and are used in our misaligned experiments.

**Estimation function families.** Following the prior work lessons [27, 45], we choose the 1-degree polynomial function family ($\{ax + b | a, b \in R\}$) and the Gaussian function family ($\{a \exp\left(\frac{-(x-b)^2}{2c^2}\right) + d | a, b, c, d \in R\}$) as representatives to compress the time series. Notice that the Gaussian function family is in ANY, while the polynomial function family is in LSF (also in VS). Table 6 summarizes the number of coefficients and error measures stored for each segment compressed by the corresponding estimation functions.

**Queries** We evaluate the correlation TSA over all the time series pairs in each dataset. The corresponding SQL queries are shown in Appendix I [6]. All the error guarantees and true errors reported in the following are the average values (including the standard variances) across all correlations in a dataset.

[8]The number of error measures for Polynomial functions is 1 since both $\|f_T\|_2$ and$\gamma_T$ can be elimated. $\|f_T\|_2$ can be elimated as Polynomial funcitons are in LSF. $\gamma_T$ can be removed as it is 0.

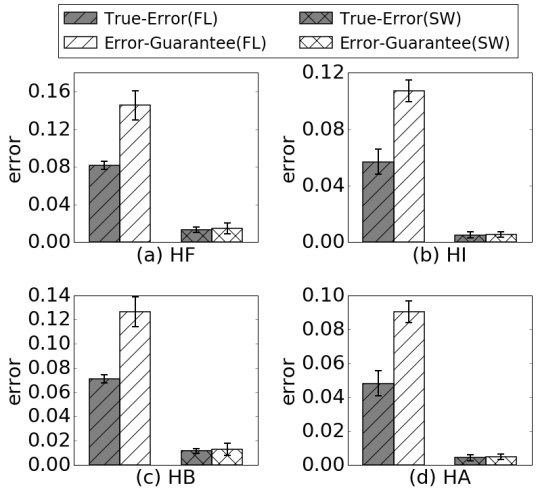



**Figure 9: True errors and error guarantees in aligned (FL) and misaligned cases (SW). The True-Error(SW) are 0.0132 and 0.00508 in (a) and (b).**

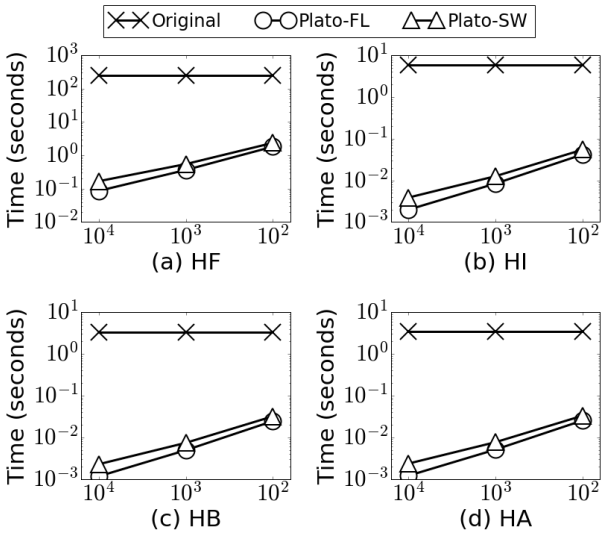



**Figure 10: Runtime of TSAs in aligned and misaligned cases as a function of compression ratio.**

## 5.2 Experimental Results

We evaluate the error guarantees for TSAs over aligned, fixed-length time series segmentations and also misaligned, variable-length time series segmentations. In order to provide a fair comparison, we fix the space cost for both cases, i.e., they have the same compression ratios.

**Performance of compression algorithms** Table 7 reports the compression time for two different compression algorithms, i.e., FL and SW [9]. To make sure the segment lists produced by SW and FL have the same compression ratios, we created segment lists as follows: (i) We first run SW for each time series. Let $M$ be the number of segments created by SW; (ii) Then we run FL by setting the segment sizes to be $N/M$ where $N$ is the number of data points in the original time series. In this way, the compression ratios in both cases are the same. The compression ratio is $\frac{cM}{N}$ where $c$ is the total number of the coefficients and the error measures. As shown both FL and SW are efficient, which indicates that compressing time series can be done online. The result is consistent with [28]. FL is slightly faster than SW, but as we show later (in Figure 9), the error guarantees produced by SW is $10\times \sim 20\times$ smaller than those produced by FL.

**Error Guarantees Quality** Figure 9 reports the absolute true errors and the error guarantees of the correlation TSAs in the aligned/fixed-length (FL) and misaligned/variable-length (SW) cases using the polynomial function family. Since the TSAs are correlations, the approximate results may range between 1 (perfect correlation) and -1 (perfect reverse correlation), with 0 meaning no correlation at all.

Under the same compression ratio [10] the variable-length error guarantees are much smaller than the fixed-length error guarantees. In Figure 9, the misaligned Error-Guarantee (SW) is $10\times \sim 20\times$ smaller than the aligned Error-Guarantee (FL) on the average (fixing the compression ratio to 1000). This is mainly because variable-length allows for much better estimation. Indeed, notice the misaligned true errors are also much smaller than the aligned true errors. For example, In Figure 9, True-Error(SW) is $6\times \sim 11\times$ smaller than True-Error(FL) on the average.

Importantly, the error guarantees are close to the true errors, especially for the misaligned error guarantees, which matter most practically. In particular, Error-Guarantee(SW) is only $1.08\times \sim 1.11\times$ larger than the True-Error(SW) in HF and HI respectively (on the average). Furthermore, they are very small in absolute terms. This indicates the high quality and practicality of AI (Amplitude-independent) error guarantees.

**Run time performance** Figure 10 reports the total running time of the correlation TSAs over (i) the original time series (Original), (ii) the time series segmented into a fixed length, aligned segments (Plato-FL) and (iii) time series segmented into misaligned, variable-length segments by SW (Plato-SW). The estimation function family is the polynomial family. The x-axis is the compression ratio (from 10000 to 100). Both Plato-FL and Plato-SW outperform vastly the Original in all the datasets. For example, when the compression ratio is 1000, Plato-FL and Plato-SW are about three orders of magnitude faster than Original. Plato-SW is about $1.8\times$ slower than Plato-FL due to the intricacy of the segment combination selection algorithm. However, a mere 80% penalty is a minor price to pay for the orders-of-magnitude superior error guarantees delivered by misaligned/variable-length segmentations.

**Comparison with sampling** In this part, we compare (i) the space cost and (ii) the runtime performance of Plato with the sampling methods when providing similar error guarantees. We use a uniform random sampling scheme with a global seed in order to create a sample database. We also assume knowledge of minimums and maximums. That is, let $X_1, ..., X_n$ be the random variables such that $d_{min} \leq X_i \leq d_{max}$ for all $i$ where $X_i = d_i^{T_1} \times d_i^{T_2}$, $d_{min} = \min\{d_i^{T_1}\} \times \min\{d_i^{T_2}\}$, and $d_{max} = \max\{d_i^{T_1}\} \times \max\{d_i^{T_2}\}$. Let $R = \sum_{i=1}^{n} X_i$ and $\varepsilon$ be the error guarantee. Using the Chernoff bounds [24], we can obtain the minimal sample size needed in order to achieve the desired error guarantee with certain confidence.

[9] SW uses the incremental update optimization.

[10] Compression ratio is the size of the original data over the size of the compressed data.

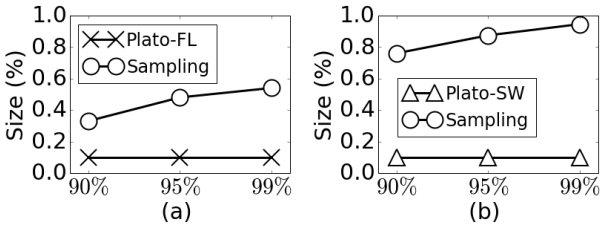



**Figure 11: Space cost of sampling and Plato when providing the same error guarantees.**

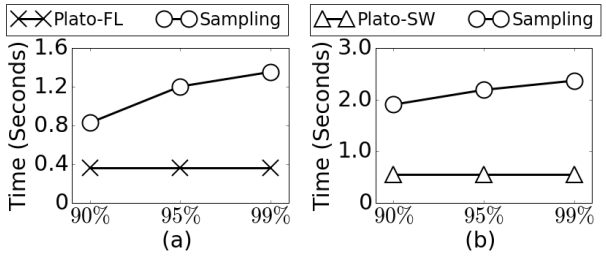



**Figure 12: Runtime of sampling and Plato when providing the same error guarantees.**

**Table 8: True errors and error guarantees for various TSAs.**

| TSA | TrueError | Plato | $\frac{\lvert Plato - TrueError\rvert}{TrueError}$ |
|---|---|---|---|
| Average | 0.00 | 0.00 | 0.00% |
| Standard Deviation | 0.00842 | 0.00842 | 0.00% |
| Cross-correlation | 0.01409 | 0.01417 | 0.56% |
| Auto-correlation | 0.00643 | 0.00645 | 0.31% |

Figure 11 reports the sizes (as percentage to the original data size) of sampled data points in order to provide similar error guarantees with the Plato-FL (the error guarantee of TSAs over aligned, fixed-length time series produced by FL) and Plato-SW (the error guarantee of TSAs over misaligned time series produced by SW) with 1000 compression ratio in HF respectively. Figure 12 shows the corresponding runtime cost. To achieve similar error guarantees, sampling needs more space and more time than Plato. We define "similar" to mean 90%, or 95% or 99% confidence - in contrast to Plato's deterministic, 100% confidence guarantees.

**Evaluation of different TSAs.** Table 8 reports the true errors (TrueError) and the error guarantees provided by Plato (Plato) and also the ratios (in the third column) for different TSAs including the average, the standard deviation, the cross-correlation and the auto-correlation in HF dataset. As shown, for TSAs on single time series, i.e., "Average" and "Standard Deviation", Plato produces the same error guarantees to the true errors (i.e., the ratios are 0.00%) as Plato can directly use the stored error measures $\gamma_T$ and $\|\varepsilon_T\|_2$ respectively. Even for TSAs over multiple time series, i.e., "Cross-correlation" and " Auto-correlation" here, Plato produces very close error guarantees to the true errors, saying the ratios are all with 1%, which verifies the high-quality of the error guarantees provided by Plato.

### 5.2.1 *Effects of Individual Factors*

In this part, we study the effects of (i) compression ratios, (ii) estimation function families, (iii) orthogonal optimizations, and (iv) segment combination selection strategies.

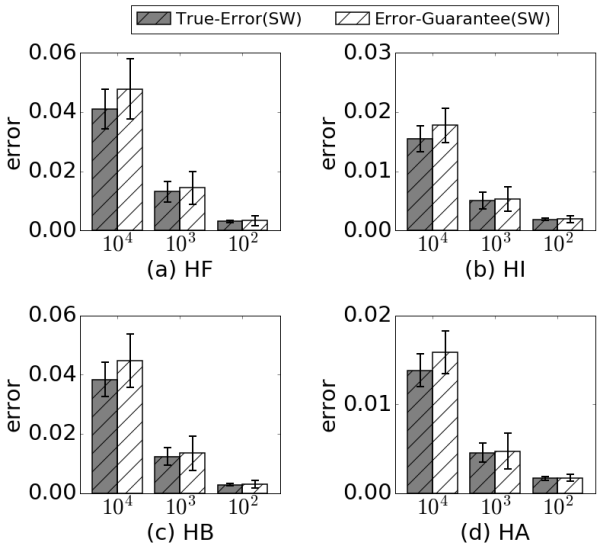



**Figure 13: Effect of compression ratios.**

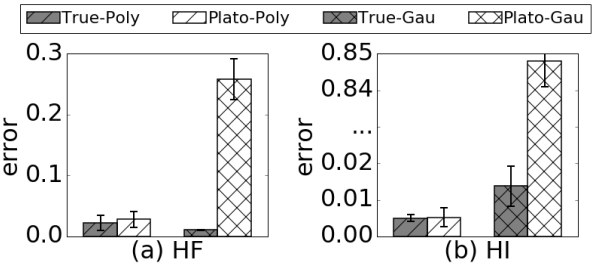



**Figure 14: Effect of estimation function families.**

**Compression ratios** In order to isolate the effect of the compression ratios, [11] we fix the estimation function family to be polynomials and fix the segment list building algorithm to be SW. In Figure 13, we change the compression ratios from 10,000 to 100 by controlling the error threshold values and report the corresponding true errors (True-Error(SW)) and the error guarantees (Error-Guarantee(SW)).

Naturally, higher compression ratios lead to smaller true errors and error guarantees. For example, in Figure 13(a), the true error and error guarantee with 100 compression ratio are 13.32× and 15.58× smaller than those with 10,000 compression ratio on the average. Importantly, the error guarantees provided by Plato are close to the true error in all the datasets and are generally small in absolute terms (with the relative exception of 10,000 compression on HF). Again, this indicates the high quality of the error guarantees provided by Plato.

**Estimation function families** In order to isolate the effect of the estimation function families, we fix the segmentation algorithm to be SW and fix the compression ratio to 1000. Figure 14 presents the true errors and the error guarantees for TSAs over time series compressed by polynomial functions (True-Error(Poly), Error-Guarantee(Poly)) and Gaussian functions (True-Error(Gau), Error-Guarantee(Gau)) respectively. The error guarantees with estimation functions

[11] Compression ratio is the size of the original data over the size of the compressed data.

from LSF (polynomials) are significantly smaller than those with estimation functions in ANY (Gaussians). In Figure 14(a), Error-Guarantee(Poly) (in LSF and VS) is about 10× smaller than Error-Guarantee(Gau) (in ANY) on the average and in Figure 14(b), Error-Guarantee(Poly) (in LSF and VS) is about 160× smaller than Error-Guarantee(Gau) (in ANY) on the average. Notice that the error guarantees provided by Plato-Poly is AI, while those of Plato-Gau are not. So the results show that AI error guarantees are practical while non-AI error guarantees are not. Interestingly, True-Error(Gau) is smaller than True-Error(Poly) in the HF dataset, which indicates that Gaussian functions model HF data better than the polynomial functions - not surprising given the more random movements of financial data. The guarantees produced by the polynomials are far better thanks to AI.

**Effect of Orthogonal Optimization and LSF** To measure the effect on error guarantees of the orthogonal optimization (and its extension to misaligned segmentations, enabled by LSF) we fix the estimation function family to the polynomials, which are LSF and, trivially, are also in ANY. We use both the general error guarantees of ANY (Error-Guarantee(ANY)) and the specialized error guarantees of LSF (Error-Guarantee(LSF)) for TSAs over misaligned segments compressed by polynomial functions (using variable-length segmentations with the SW algorithm). We fix the compression ratio to 1000. As shown in Figure 15, the error guarantee for LSF certifies that the true result is just within ±0.0137 in HF and within ±0.0052 in HI.

**Segment combination selection strategies** To isolate the quality effect of employing the optimal segment combination selection strategy (OS) we compare it with IS strategy (the straightforwad method mentioned in Section 4.4.1) on a case of variable-length compression with an LSF function family (polynomials). Figure 16 shows that Plato-OS is about 5× smaller than Plato-MS on the average. In addition, the runtime of Plato-IS and Plato-OS are close. For example, the running time of Plato-IS and Plato-OS are 0.536 and 0.548 seconds in HF respectively.

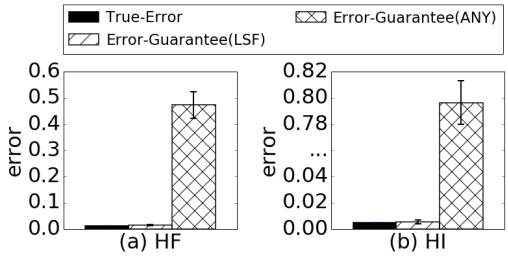



**Figure 15: Effect of orthogonal optimization.**

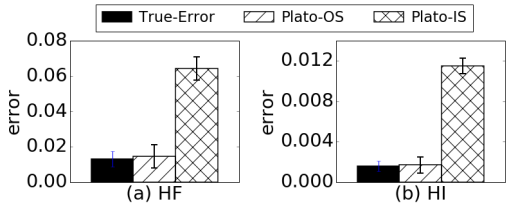



**Figure 16: Effect of segment combination selection strategies.**

# 6. RELATED WORK

**AQP with probabilistic error guarantees.** Approximate query processing using *sampling* [8, 54, 46, 4] computes approximate answers by appropriately evaluating the queries on small samples of the data, e.g., STRAT [8], SciBORQ [54], BlinkDB [4], and Druid[1]. In particular, the BlinkDB shows the probabilistic error guarantees in the user interface. Such approaches typically leverage statistical inequalities and the central limit theorem to compute the confidence interval (or variance) of the computed approximate answer. As a result, their error guarantees are probabilistic - as opposed to this work's deterministic (100% confidence) ones. Note however that, unlike sampling, our compression-based techniques are tuned for time series and continuous data.

**AQP with deterministic error guarantees.** Approximately answering queries while providing deterministic error guarantees has been successfully applied in many applications [13, 23, 40, 51, 36, 50]. However, existing work in the area has focused on simple aggregation queries that involve only a single time series (or table) and aggregates such as SUM, COUNT, MIN, MAX and AVG. Our work extends the prior work, as it addresses analytics over multiple compressed time series such as correlation, cross-correlation.

**Data summarizations and compressions** The database community has mostly focused on creating summarizations (also referred to as synopses or sketches) that can be used to answer specific queries. These include among others histograms [49, 25, 57, 53] (e.g., EquiWidth and EquiDepth histograms [49], V-Optimal histograms [25], and Hierarchical Model Fitting (HMF) histograms [57]), used among other for cardinality estimation [25] and selectivity estimation [50]. The signal processing community produced a variety of methods that can be used to compress time series data and thus are more relevant to the present work, as they provide the underlying compressions. These include among others the Piecewise Aggregate Approximation (PAA) [30], and the Piecewise Linear Representation (PLR) [27], and the new piecewise compression technique proposed in [16]. Plato is orthogonal to those data summarization and compression techniques.

**Other time series management system** There are existing time series management systems sovling orthogonal problems. For example ModelarDB [26] focuses on applying models to store sensor data; [52] proposes an adaptive algorithm to choose the most compact function to compress time series; [41] studies similarity queries over compressed time series; and both CrateDB[2] and Timescale[3] make SQL scalable for time-series data. They are orthogonal to Plato as Plato focuses on providing error guarantees for online analytic queries over multiple compressed time series.

# 7. SUMMARY AND FUTURE DIRECTION

This work indicates that deterministic error guarantees are feasible and practical, given the appropriate combination of error measures and estimation function family. Future work may develop such combinations for other important families also. Note that the tightness results of this paper do not preclude the future development of practical and theoretically-sound deterministic error guarantees for families that are currently outside the LSF. Researchers may come up with other interesting properties of function families outside LSF (or VS) and deliver good error guarantees.

# 8. REFERENCES


[1] https://druid.apache.org/.

[2] https://crate.io/.

[3] https://www.timescale.com/.

[4] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *EuroSys*, pages 29–42, 2013.

[5] S. R. Aghabozorgi, A. S. Shirkhorshidi, and Y. W. Teh. Time-series clustering - A decade review. *Inf. Syst.*, 53:16–38, 2015.

[6] E. Boursier, J. J. Brito, C. Lin, and Y. Papakonstantinou. Plato: Approximate analytics over compressed time series with tight deterministic error guarantees. *CoRR*, abs/1808.04876, 2018.

[7] K. Chan and A. W. Fu. Efficient time series matching by wavelets. In *ICDE*, pages 126–133, 1999.

[8] S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. *TODS*, 32(2):9, 2007.

[9] S. Chaudhuri, B. Ding, and S. Kandula. Approximate query processing: no silver bullet. In *Sigmod*, pages 511–519. ACM, 2017.

[10] L. Chen and R. T. Ng. On the marriage of lp-norms and edit distance. In *VLDB*, pages 792–803, 2004.

[11] W. Cheney and D. Kincaid. Linear algebra: Theory and applications. *The Australian Mathematical Society*, 110, 2009.

[12] B. Choi. *ARMA model identification*. Springer Science & Business Media, 2012.

[13] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Effective computation of biased quantiles over data streams. In *ICDE*, pages 20–31, 2005.

[14] G. Das, K. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. In *KDD*, pages 16–22, 1998.

[15] D. Denison, B. Mallick, and A. Smith. Automatic bayesian curve fitting. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 60(2):333–350, 1998.

[16] F. Eichinger, P. Efros, S. Karnouskos, and K. Böhm. A time-series compression technique and its application to the smart grid. *VLDB J.*, 24(2):193–218, 2015.

[17] A. Eisenberg and J. Melton. Sql/xml is making good progress. *ACM Sigmod Record*, 31(2):101–108, 2002.

[18] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, pages 419–429, 1994.

[19] T.-c. Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011.

[20] T.-c. Fu, F.-l. Chung, V. Ng, and R. Luk. Evolutionary segmentation of financial time series into subsequences. In *Proceedings of the 2001 Congress on Evolutionary Computation*, volume 1, pages 426–430, 2001.

[21] A. Galakatos, A. Crotty, E. Zgraggen, C. Binnig, and T. Kraska. Revisiting reuse for approximate query processing. *PVLDB*, 10(10):1142–1153, 2017.

[22] A. Galakatos, M. Markovitch, C. Binnig, R. Fonseca, and T. Kraska. Fiting-tree: A data-aware index structure. In *SIGMOD*, pages 1189–1206, 2019.

[23] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD*, pages 58–66, 2001.

[24] T. Hagerup and C. Rüb. A guided tour of chernoff bounds. *Information processing letters*, 33(6):305–308, 1990.

[25] Y. E. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *SIGMOD*, pages 233–244, 1995.

[26] S. K. Jensen, T. B. Pedersen, and C. Thomsen. Modelardb: Modular model-based time series management with spark and cassandra. *PVLDB*, 11(11):1688–1701, 2018.

[27] E. Keogh. Fast similarity search in the presence of longitudinal scaling in time series databases. In *ICTAI*, pages 578–584, 1997.

[28] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *ICDM*, pages 289–296, 2001.

[29] E. Keogh, S. Chu, D. Hart, and M. Pazzani. Segmenting time series: A survey and novel approach. In *Data mining in time series databases*, pages 1–21. World Scientific, 2004.

[30] E. J. Keogh, K. Chakrabarti, M. J. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *KAIS*, 3(3):263–286, 2001.

[31] E. J. Keogh and M. J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *KDD*, pages 239–243, 1998.

[32] E. J. Keogh and M. J. Pazzani. Relevance feedback retrieval of time series data. In *SIGIR*, pages 183–190, 1999.

[33] K. Kim. Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1-2):307–319, 2003.

[34] A. Koski, M. Juhola, and M. Meriste. Syntactic recognition of ecg signals by attributed finite automata. *Pattern Recognition*, 28(12):1927–1940, 1995.

[35] G. Kovács, S. Zucker, and T. Mazeh. A box-fitting algorithm in the search for periodic transits. *Astronomy & Astrophysics*, 391(1):369–377, 2002.

[36] I. Lazaridis and S. Mehrotra. Progressive approximate aggregate queries with a multi-resolution tree structure. In *SIGMOD*, pages 401–412, 2001.

[37] I. Lazaridis and S. Mehrotra. Capturing sensor-generated time series with quality guarantees. In *ICDE*, pages 429–440, 2003.

[38] C. Li, P. S. Yu, and V. Castelli. MALM: A framework for mining sequence database at multiple abstraction levels. In *CIKM*, pages 267–272, 1998.

[39] X. Liu, Z. Lin, and H. Wang. Novel online methods for time series segmentation. *TKDE*, 20(12):1616–1626, 2008.

[40] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *SIGMOD*, pages 426–435, 1998.

[41] A. Marascu, P. Pompey, E. Bouillet, M. Wurst, O. Verscheure, M. Grund, and P. Cudré-Mauroux. TRISTAN: real-time analytics on massive time series using sparse dictionary compression. In *Big Data*, pages 291–300, 2014.

[42] J. Mei and J. M. F. Moura. Signal processing on graphs: Causal modeling of unstructured data. *IEEE Trans. Signal Processing*, 65(8):2077–2092, 2017.

[43] M. D. Morse and J. M. Patel. An efficient and accurate method for evaluating time series similarity. In *SIGMOD*, pages 569–580, 2007.

[44] E. Nelson. Probability theory and euclidean field theory. In *Constructive quantum field theory*, pages 94–124. Springer, 1973.

[45] Z. Pan, Y. Hu, and B. Cao. Construction of smooth daily remote sensing time series data: a higher spatiotemporal resolution perspective. *Open Geospatial Data, Software and Standards*, 2(1):25, 2017.

[46] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie. Online aggregation for large mapreduce jobs. *PVLDB*, 4(11):1135–1145, 2011.

[47] S. Park, D. Lee, and W. W. Chu. Fast retrieval of similar subsequences in long sequence databases. In *KDEX*, pages 60–67, 1999.

[48] J. S. Philo. An improved function for fitting sedimentation velocity data for low-molecular-weight solutes. *Biophysical Journal*, 72(1):435–444, 1997.

[49] G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. In *SIGMOD*, pages 256–276, 1984.

[50] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In *SIGMOD*, pages 294–305, 1996.

[51] N. Potti and J. M. Patel. DAQ: A new paradigm for approximate query processing. *PVLDB*, 8(9):898–909, 2015.

[52] J. Qi, R. Zhang, K. Ramamohanarao, H. Wang, Z. Wen, and D. Wu. Indexable online time series segmentation with error bound guarantee. *World Wide Web*, 18(2):359–401, 2015.

[53] F. Reiss, M. N. Garofalakis, and J. M. Hellerstein. Compact histograms for hierarchical identifiers. In *VLDB*, pages 870–881, 2006.

[54] L. Sidirourgos, M. L. Kersten, and P. A. Boncz. Sciborq: Scientific data management with bounds on runtime and quality. In *CIDR*, pages 296–301, 2011.

[55] M. Tobita. Combined logarithmic and exponential function model for fitting postseismic gnss time series after 2011 tohoku-oki earthquake. *Earth, Planets and Space*, 68(1):41, 2016.

[56] M. Vlachos, D. Gunopulos, and G. Kollios. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684, 2002.

[57] H. Wang and K. C. Sevcik. Histograms based on the minimum description length principle. *VLDB J.*, 17(3):419–442, 2008.

[58] W. Wiscombe and J. Evans. Exponential-sum fitting of radiative transmission functions. *Journal of Computational Physics*, 24(4):416–444, 1977.