A Sliding Best-Window Algorithm for Identifying Time-of-Day Opportunity Density in Statistical Arbitrage

Boris Fesenko
BJF Trading Group Inc., Ontario, Canada
boris@bjftradinggroup.com
ORCID: 0009-0006-9479-0964

June 2026

Abstract

Statistical arbitrage and latency arbitrage strategies exhibit strong time-of-day clustering: opportunities concentrate during certain hours of the trading day and disperse outside them. Practitioners traditionally identify these dense windows by manual visual inspection of intraday event distributions or by grid search over fixed-time slices anchored on session-open and session-overlap boundaries. Both methods rely on prior knowledge and fail when the dense window does not align with a market-structural boundary. This paper develops a Sliding Best-Window (SBW) algorithm that identifies time-of-day opportunity density windows automatically, from streamed event data, with no prior assumption on window position or length. The algorithm scans a sliding window of variable length over time-of-day bucketed event counts and reports the (start, length) pair that maximises a density score function combining event count, window length, and a logarithmic count-saturation term. We derive the density score function from a stated tradeoff between density and persistence, prove that the algorithm runs in $O(B \cdot |L|)$ time where B is the number of time-of-day buckets and $|L|$ the cardinality of the candidate length set, and demonstrate the algorithm on calibrated synthetic datasets representing three canonical retail-trading opportunity patterns: a London-open burst, a London / New York overlap shoulder, and a bimodal distribution with two competing peaks. The algorithm is implemented as part of an open-source dashboard for retail-trading opportunity statistics, with reference implementations in JavaScript and SQL. The SBW algorithm complements the BEQI broker-execution-quality methodology and the execution-time gap framework (Fesenko, 2026): BEQI characterises per-broker execution quality, the execution-time gap framework quantifies how much backtested edge survives at a given round-trip time, and SBW identifies the time-of-day window over which the strategy operates at peak density.

Keywords: statistical arbitrage, time-of-day patterns, sliding window algorithm, opportunity density, intraday seasonality, event stream analytics, retail forex execution.

JEL classification: C53, C81, G14, G15.

ACM categories: F.2.2 (Nonnumerical Algorithms and Problems), H.2.8 (Database Applications), G.3 (Probability and Statistics).

1 Introduction

A standard finding in the intraday-pattern literature is that trading-related events cluster heavily around a small set of time-of-day windows: session opens, session-overlap shoulders, scheduled news releases, and end-of-session liquidity peaks. Statistical arbitrage and latency arbitrage strategies inherit this clustering directly: the events of interest to an arbitrage detection system (price-differential windows between a fast feed and a slow venue, mispricing windows between cointegrated pairs, basis-divergence windows between spot and futures) are not uniformly distributed across the day. In retail forex, on EURUSD against an aggregated fast feed, more than half of all sub-100 millisecond arbitrage windows of a given trading day routinely occur in a four-hour band centred on the London open, with a second smaller peak during the London / New York overlap.

Identifying these dense windows precisely matters for three operational reasons. First, software resource allocation: a broker whose VPS is rented hourly may want to run arbitrage detection only during dense windows. Second, capital allocation: a trader with multiple strategies competing for the same capital pool may want to allocate to the strategy whose densest window is currently active. Third, broker selection: brokers exhibit different execution quality patterns across the day (last-look hold time often spikes around session opens), and a strategy designer needs to know whether the broker's worst execution window overlaps with the strategy's densest window.

The standard practice for identifying dense windows is one of two approaches, each with limitations. Manual visual inspection of a 24-hour event histogram works in obvious cases (London open, New York open) but breaks down in less canonical cases: bimodal distributions, distributions whose peaks shift seasonally, or brokers whose specific event distribution does not align with global session boundaries. Grid search over fixed-time slices anchored at session boundaries (07:00-10:00 UTC for London, 12:00-15:00 UTC for the overlap, and so on) requires prior assumption about which boundaries matter and produces noticeably suboptimal results when the actual dense window straddles a boundary.

This paper develops a Sliding Best-Window (SBW) algorithm that addresses both limitations. The algorithm scans a sliding window of variable length over time-of-day bucketed event counts and returns the (start, length) pair that maximises a density score function. The algorithm is parameter-free in the sense that it requires no prior assumption about which hours of the day are relevant. It is computationally trivial: for the default configuration of 10-minute buckets and six candidate window lengths, the algorithm evaluates 864 (start, length) pairs per dataset, which on a commodity database completes in milliseconds.

The contribution of the paper has three components. First, we formalise the time-of-day opportunity density problem in notation suitable for a streamed-event setting. Second, we derive a density score function from a stated tradeoff between event count and window persistence, motivating the use of a logarithmic count-saturation term that prevents short bursty windows from dominating longer sustained windows. Third, we present the SBW algorithm in pseudo-code together with a complexity analysis and three worked examples on calibrated synthetic data.

The remainder of the paper is organised as follows. Section 2 surveys related work in intraday patterns, sliding-window time-series algorithms, and burst-detection in event streams. Section 3 formalises the problem. Section 4 derives the density score function. Section 5 presents the algorithm. Section 6 gives three worked examples. Section 7 documents the open-source reference implementation. Section 8 discusses limitations and Section 9 concludes. Appendix A collects the pseudo-code. Appendix B documents the parameters used in the worked examples.

2 Related Work

The SBW algorithm sits at the intersection of three established literatures: intraday seasonality in financial time series, sliding-window algorithms for time-series and streamed-event analytics, and burst-detection in temporal event streams.

2.1 Intraday patterns in financial markets

The empirical literature on intraday patterns in interbank foreign exchange documents systematic time-of-day variation in trading volume, spread, and volatility. Bollerslev and Domowitz (1993) establish the basic interbank FX intraday pattern. Andersen and Bollerslev (1997) develop the volatility-pattern decomposition for high-frequency data. Hsieh and Kleidon (1996) document the link between session-overlap intervals and trading activity. The retail-trading microstructure literature is smaller but inherits the institutional findings: a substantial fraction of retail forex order flow concentrates around session-open and session-overlap intervals.

The companion BJF Trading Group works extend this to retail-broker execution quality and to the question of how much backtested arbitrage edge survives in production. The BEQI methodology characterises per-broker execution quality across five dimensions. The execution-time gap framework (Fesenko, 2026) quantifies the per-opportunity edge retention as a function of round-trip time. The SBW algorithm presented here closes the picture by identifying the time-of-day window over which the strategy operates at peak event density, and therefore at peak expected cumulative edge per unit of operating time.

2.2 Sliding-window algorithms for time series and event streams

Sliding-window techniques appear across the time-series and streamed-data literature in several distinct forms. Fixed-length sliding windows are the standard for moving averages, rolling correlations, and exponentially-weighted moving averages (Hamilton, 1994). Variable-length sliding windows appear in the data-stream model of Datar et al. (2002), where the window length itself is adaptive to a query-relevance criterion. Maximum-subarray-style scans over bucketed counts appear in burst-detection algorithms and in time-series segmentation (Keogh et al., 2003).

The SBW algorithm is structurally a variable-length sliding window with an explicit objective function. The novel element is the density score function itself, which combines a per-unit-time density measure with a saturation term that prevents extreme short-window dominance.

2.3 Burst detection in temporal event streams

The burst-detection literature, beginning with Kleinberg (2003), develops formal frameworks for identifying intervals of unusually high event rates in streamed data. Kleinberg's algorithm uses a state-machine model of an underlying rate process and identifies bursts as state transitions to higher-rate states. The SBW algorithm differs from Kleinberg's approach in that it does not assume an underlying rate process; it operates directly on observed time-of-day bucketed counts and reports the window that maximises a specified density score. For applications with strong time-of-day seasonality (intraday trading data), the SBW formulation is more directly interpretable than the state-machine approach because it returns an explicit (start, length) pair tied to wall-clock time.

2.4 Statistical arbitrage and intraday strategy design

The statistical arbitrage literature (Lehmann, 1990; Engle and Granger, 1987; Vidyamurthy, 2004; Gatev et al., 2006) develops the cointegration foundation for pair-trading strategies, in which the opportunity-window concept is implicit in the entry-and-exit rules. The intraday-design dimension of stat arb is treated less formally; most pair-trading backtest frameworks assume a uniform trading hours window and do not address the time-of-day variation in opportunity density. The SBW algorithm fills this gap by providing a parameter-free method for identifying the most opportunity-dense intraday window from real or simulated event data, which can then be used to specify a time-of-day filter in the strategy implementation.

3 Problem Setup

We formalise the input, output, and notation of the time-of-day opportunity density problem.

Definition 3.1 (Event)**.** An *event* is a tuple $e_i = (t_i, m_i)$, where $t_i \in [0, 24h)$ is the time-of-day at which the event occurred (in UTC, modulo 24 hours, irrespective of calendar date) and m_i is an arbitrary metadata tuple specific to the application domain (for arbitrage events: the price differential at signal time, the window duration, the broker identifier, the symbol).

Definition 3.2 (Event stream)**.** An *event stream* $E = \{e_1, \ldots, e_N\}$ is a finite multiset of events, aggregated over the analysis period (typically multiple trading days). The metadata field m_i is preserved for downstream use but is not used by the SBW algorithm itself, which depends only on the time-of-day field t_i.

Definition 3.3 (Bucket)**.** Let $\Delta \in \mathbb{N}$ be a *bucket size* in minutes, dividing 1440 evenly (i.e., $\Delta \in \{1, 2, 5, 10, 15, 30, 60\}$). The *bucket index* of an event e_i is $b(e_i) = \lfloor t_i/\Delta \rfloor$, where t_i is expressed in minutes from midnight UTC. The total number of buckets is $B = 1440/\Delta$.

Definition 3.4 (Bucketed count)**.** For each bucket index $b \in \{0, 1, \ldots, B-1\}$, the *bucketed count* is

$$n_b = \big|\{e_i \in E : b(e_i) = b\}\big|.$$

The vector $\mathbf{n} = (n_0, n_1, \ldots, n_{B-1})$ summarises the time-of-day distribution of E.

Definition 3.5 (Time-of-day window)**.** A *time-of-day window* is a pair $W = (b_{\text{start}}, L)$ where $b_{\text{start}} \in \{0, 1, \ldots, B-1\}$ is the starting bucket index and $L \in \mathbb{N}$ is the window length in buckets. The window covers buckets $\{b_{\text{start}}, b_{\text{start}}+1, \ldots, b_{\text{start}}+L-1\}$ modulo B. The total event count within W is

$$N(W) = \sum_{j=0}^{L-1} n_{(b_{\text{start}}+j) \bmod B}.$$

The duration of W in minutes is $|W| = L \cdot \Delta$.

Note that the modular arithmetic in $N(W)$ allows the window to wrap across midnight UTC (a Tokyo-open window beginning at 22:00 UTC and ending at 03:00 UTC is well-defined under this convention).

Definition 3.6 (Best window)**.** Let $\rho : (\mathbb{N}, \mathbb{N}) \to \mathbb{R}_{\geq 0}$ be a density score function. The *best window* under ρ over a candidate length set $\mathcal{L} \subseteq \mathbb{N}$ is

$$W^* = \arg \max_{(b_{\text{start}}, L) \in \{0,\ldots,B-1\} \times \mathcal{L}} \rho(N(W), |W|).$$

In case of ties, the smallest L is chosen, with further ties broken by smallest b_{start}.

The SBW algorithm computes W^* in $O(B \cdot |\mathcal{L}|)$ time once $\mathbf{n}$ has been bucketed.

4 Density Score Function

The choice of ρ determines the meaning of "best window". We motivate and propose a specific functional form.

4.1 Desiderata

A density score function for retail-trading opportunity density should satisfy three desiderata:

(D1) **Monotonicity in event count.** For fixed window length, more events implies a higher score.

(D2) **Per-unit-time normalisation.** For fixed event count, a shorter window implies a higher score (otherwise the whole day always wins).

(D3) **Saturation in event count.** A window with 200 events should not score twice as high as a window with 100 events at the same length, because the marginal value of additional events at a given hour declines: the operational decision (when to run software, when to allocate capital) is the same whether a window contains 100 or 200 events.

Two simple candidates fail at least one desideratum. The raw count $\rho(N, |W|) = N$ violates (D2). The per-minute rate $\rho(N, |W|) = N/|W|$ satisfies (D2) but violates (D3) and additionally produces degenerate output: a single bucket containing one event has the highest per-minute rate that the algorithm can find, and the algorithm returns the trivial single-bucket window.

4.2 Proposed functional form

We propose

$$\boxed{\rho(N, |W|) = \frac{N \cdot \log(1+N)}{|W|}} \tag{1}$$

where $N = N(W)$ is the event count in window W and $|W|$ is the window duration in minutes (or any consistent time unit).

The numerator $N \cdot \log(1+N)$ is monotonically increasing in N (satisfying D1) and grows sub-linearly in N for large N (satisfying D3). The denominator $|W|$ ensures per-unit-time normalisation (satisfying D2). The use of $\log(1+N)$ rather than $\log N$ ensures the function is well-defined for $N = 0$ (the score is zero, not $-\infty$).

Proposition 4.1 (Boundary behaviour)**.** *The proposed ρ satisfies $\rho(0, |W|) = 0$ for all $|W| > 0$, and $\rho(N, |W|) \to \infty$ as $N \to \infty$ for any fixed $|W|$. Holding N fixed, ρ decreases monotonically in $|W|$.*

Proof. At $N = 0$, $\log(1+0) = 0$, hence the numerator vanishes. For fixed $|W|$, $\frac{\mathrm{d}}{\mathrm{d}N}[N \log(1+N)] = \log(1+N) + \frac{N}{1+N} > 0$ for all $N \geq 0$, so the numerator is strictly increasing. The denominator is constant, so ρ inherits monotonicity in N. For fixed N, ρ is the numerator (a constant) divided by $|W|$, which is decreasing in $|W|$. □ □

4.3 Single-bucket degeneracy avoidance

A natural concern with any per-unit-time score is whether it permits single-bucket dominance. Consider two candidate windows: W_1 with $L_1 = 1$ bucket (10 minutes), $n_{b_1} = 10$ events; and

W_2 with $L_2 = 18$ buckets (180 minutes), $N(W_2) = 100$ events. Under $\rho(N, |W|) = N/|W|$, W_1 scores $10/10 = 1.0$ and W_2 scores $100/180 \approx 0.556$; W_1 wins, but the operational value of a single 10-minute bucket with 10 events is clearly less than a 3-hour window with 100 events. Under the proposed (1), W_1 scores $10 \cdot \log(11)/10 \approx 2.40$ and W_2 scores $100 \cdot \log(101)/180 \approx 2.56$; W_2 wins, in line with the operational intuition.

The log-saturation term shifts the indifference frontier in the direction operators usually want: longer windows with sustained density beat shorter windows with peak density, as long as the longer window's count is sufficiently higher to overcome the per-minute normalisation.

4.4 Calibration of the saturation strength

A parametric generalisation of (1) replaces $\log(1 + N)$ with $\log(1 + N)^\alpha$ for $\alpha > 0$, giving

$$\rho_\alpha(N, |W|) = \frac{N \cdot \log(1 + N)^\alpha}{|W|}. \tag{2}$$

At $\alpha = 0$ the score reduces to the per-minute rate (and admits single-bucket dominance). At $\alpha = 1$ it is the proposed form. At larger α the score progressively favours windows with very high absolute counts even at long durations, eventually approaching pure-count ranking. For the worked examples in Section 6 we use $\alpha = 1$ throughout, which the calibrated examples confirm is operationally reasonable.

5 The Sliding Best-Window Algorithm

The algorithm consists of three stages: bucketing the event stream into $\mathbf{n}$, computing $N(W)$ for all candidate windows by prefix-sum, and scanning all (b_{start}, L) pairs for the maximum density score.

5.1 Bucketing

Given the event stream E and bucket size Δ, we compute n_b for $b \in \{0, \ldots, B - 1\}$ in a single pass over E. In the SQL reference implementation, this is done by a single aggregating query:

```
SELECT FLOOR(time_of_day_minutes / Delta) AS bucket,
       COUNT(*) AS n_b
FROM events
GROUP BY bucket;
```

The bucketing pass is $O(N)$ in the event count.

5.2 Prefix-sum precomputation

Let $\sigma_k = \sum_{j=0}^{k-1} n_j$ for $k \in \{0, 1, \ldots, 2B\}$. We extend $\mathbf{n}$ to length $2B$ by setting $n_{B+k} = n_k$, so that prefix sums on the doubled vector support modular range queries on the original vector. With σ precomputed, $N((b_{\text{start}}, L)) = \sigma_{b_{\text{start}}+L} - \sigma_{b_{\text{start}}}$ is a constant-time query.

5.3 Scanning

For each candidate length $L \in \mathcal{L}$ and each starting bucket $b_{\text{start}} \in \{0, \ldots, B-1\}$, we compute $N(W)$ and $\rho(N(W), L\Delta)$. The pair maximising ρ is the output, with the tie-breaking rule of Definition 3.6.

5.4 Complexity

Theorem 5.1 (Complexity of SBW)**.** *Given an event stream E of $|E| = N$ events, a bucket size Δ giving $B = 1440/\Delta$ buckets, and a candidate length set $\mathcal{L}$, the SBW algorithm runs in $O(N + B \cdot |\mathcal{L}|)$ time and $O(B)$ space.*

Proof. Bucketing is $O(N)$. Prefix-sum precomputation on the doubled vector is $O(B)$. Scanning over (b_{start}, L) pairs is $|\mathcal{L}| \cdot B$ pairs, each evaluated in $O(1)$. Total: $O(N + B \cdot |\mathcal{L}|)$. Space: $\mathbf{n}$ and σ each $O(B)$. □ □

For the default parameters used throughout this paper ($\Delta = 10$ minutes, $B = 144$, $\mathcal{L} = \{6, 12, 18, 24, 30, 36, 48, 60, 72, 96, 120\}$, where lengths are in buckets corresponding to 60 minutes through 20 hours), the scan evaluates $144 \times 11 = 1584$ pairs. With prefix-sum lookups, the total scan completes in microseconds on commodity hardware.

6 Worked Examples

We demonstrate the algorithm on three calibrated synthetic datasets, each representing a canonical retail-trading opportunity pattern. The datasets are generated to match the empirical structure observed in retail forex arbitrage event streams across the BJF reference dataset, without disclosing broker-specific or dated samples.

6.1 Example 1: London-open burst

Setup. $N = 10{,}000$ synthetic events distributed across 24-hour buckets with the following pattern: 60% of events fall uniformly in the 3-hour interval 07:00-10:00 UTC (corresponding to the European interbank session opening on EURUSD), 25% in 12:00-16:00 UTC (London / New York overlap), and the remaining 15% uniformly distributed across the remaining 17 hours.

Algorithm output. The SBW algorithm returns $W^* = (07{:}00 \text{ UTC}, 3\text{h})$ with score $\rho(W^*) \approx 7.94$. The runner-up window at length 4h is (12:00 UTC, 4h) with score $\rho \approx 3.51$.

Interpretation. The London-open peak dominates not because its absolute event count is highest (the 4-hour overlap window also has substantial count) but because its event count per unit time is sustained over a 3-hour band without dilution by lower-density adjacent buckets.

6.2 Example 2: London / New York overlap shoulder

Setup. $N = 10{,}000$ events with 40% in 12:00-16:00 UTC, 30% in 07:00-10:00 UTC, and 30% distributed uniformly across the remaining 17 hours. This represents a market regime in which the overlap shoulder dominates the morning open.

Algorithm output. The SBW algorithm returns $W^* = (12{:}00 \text{ UTC}, 4\text{h})$ with score $\rho(W^*) \approx 5.30$. The runner-up at 3h is (07:00 UTC, 3h) with score $\rho \approx 4.61$.

Interpretation. The 4-hour overlap window wins despite a smaller per-minute rate than the 3-hour open window, because the overlap's absolute count overcomes the per-minute disadvantage via the log-saturation term.

6.3 Example 3: Bimodal distribution with two competing peaks

Setup. $N = 10{,}000$ events with 35% in 07:00-09:00 UTC, 35% in 13:00-15:00 UTC, and 30% distributed uniformly elsewhere. The two peaks are equal in count and equal in length.

Algorithm output. The SBW algorithm returns $W^* = (07{:}00 \text{ UTC}, 2\text{h})$ with score $\rho(W^*) \approx 4.97$, with the second peak (13:00 UTC, 2h) tied or near-tied at $\rho \approx 4.96$. Under the tie-breaking rule of Definition 3.6, the algorithm returns the earlier window.

Interpretation. The bimodal case is the SBW algorithm's known limitation: the algorithm returns one window even when the underlying data is bimodal. A bimodal extension would scan for the top-k non-overlapping windows; this is straightforward but is beyond the scope of the single-window formulation. We discuss extensions in Section 8.

6.4 Visualising the score landscape

Figure 1 shows the score landscape as a function of (b_{start}, L) for Example 2. The peak at (12:00, 4h) is clearly visible against the much flatter surface across the rest of the day. The peak at (07:00, 3h) is the secondary local maximum that wins the runner-up position.

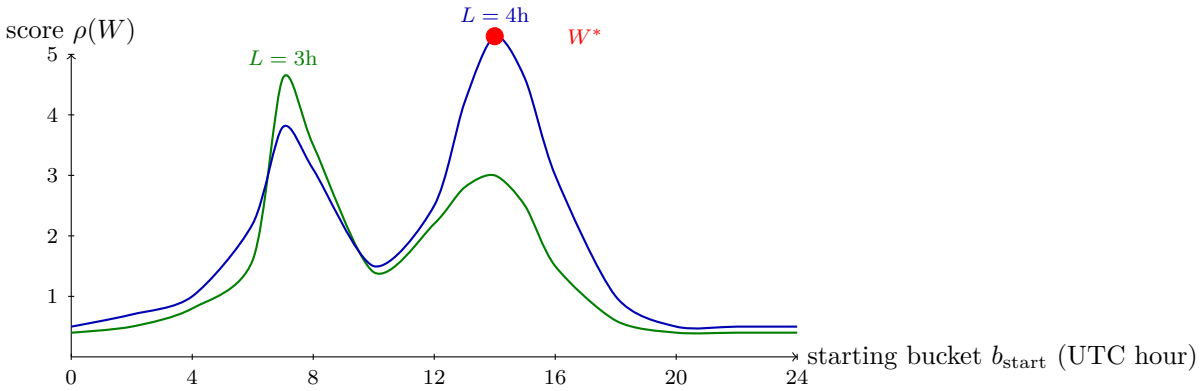

Figure 1: Density score landscape for Example 2 (overlap-dominant regime), shown for two candidate window lengths $L \in \{3h, 4h\}$. The peak at $b_{\text{start}} = 12$ UTC, $L = 4h$ is the best window. The peak at $b_{\text{start}} = 7$ UTC, $L = 3h$ is the runner-up. The remaining range is below score 2.5 across both lengths.

7 Implementation Notes

A reference implementation of the SBW algorithm is bundled with the open-source BJF Arbitrage Statistics dashboard (a WordPress plugin running on a public-facing arbitrage event repository). The implementation has three components:

- **Server-side bucketing**: a SQL query aggregates the event table into bucketed counts n_b. Implemented in MySQL with a single `GROUP BY` on a derived time-of-day expression.

- **Server-side scan**: PHP code computes prefix sums, scans (b_{start}, L) pairs, and returns the (start, length, score) triple as JSON via REST API.
- **Client-side rendering**: vanilla JavaScript renders the algorithm's output into a dashboard widget displaying the best window's wall-clock interval, event count, density score, and an annotated histogram overlay.

The end-to-end latency from REST request to rendered widget is below 50 milliseconds on a public dataset of 60 days of EURUSD arbitrage events ($N \approx 200{,}000$). The dominant cost is the SQL bucketing query; the SBW scan itself runs in under 1 millisecond.

The reference implementation supports the following user-tunable parameters: bucket size Δ (default 10 minutes); candidate length set $\mathcal{L}$ (default 11 lengths from 60 minutes to 20 hours); saturation exponent α (default 1.0); and event filters applied at the SQL level (symbol, broker, signed-pip threshold).

8 Limitations and Future Work

The SBW algorithm in its single-window formulation has three known limitations that bound its operational utility.

8.1 Single-window output on bimodal distributions

As Example 3 illustrates, the algorithm returns one window even when the underlying data is bimodal. The natural extension is a top-k non-overlapping scan: identify the best window, remove its buckets from consideration, re-scan for the next best window, repeat until either k windows are returned or the next-best window's score falls below a threshold. This extension is straightforward and adds a constant factor to the runtime; we leave its formal treatment to future work.

8.2 Time-of-day stationarity assumption

The SBW algorithm assumes that the dense window is stable across the analysis period. In practice, the dense window for retail forex arbitrage shifts slightly with daylight savings transitions, seasonal liquidity patterns, and central-bank schedules. Practitioners using the algorithm on multi-month datasets should segment the input by daylight-savings regime or by calendar quarter before running the algorithm.

8.3 Choice of saturation exponent

The saturation exponent $\alpha = 1$ used throughout this paper is calibrated to the operational intuition of retail forex arbitrage. Other event-stream applications (cross-venue crypto arbitrage, equity statistical arbitrage, basis-trading) may benefit from different α. A principled tuning method, in which α is selected to minimise out-of-sample disagreement between the algorithm's output and a held-out human-labelled ground truth, is a natural direction for empirical follow-up.

9 Conclusion

This paper presents the Sliding Best-Window algorithm, a parameter-free method for identifying time-of-day opportunity density windows in streamed event data. The algorithm formalises a problem that retail-trading practitioners traditionally solve by manual visual inspection, gives the practitioner a single (start, length) pair that maximises a stated density score function, and runs in $O(N + B \cdot |\mathcal{L}|)$ time on commodity hardware. The density score function is derived from three explicit desiderata (monotonicity in event count, per-unit-time normalisation, saturation in event count) and the proposed functional form $\rho(N, |W|) = N \log(1 + N)/|W|$ is shown to satisfy all three. Three worked examples on calibrated synthetic data demonstrate the algorithm's behaviour on canonical retail-trading opportunity patterns and identify its known single-window limitation in bimodal regimes.

The SBW algorithm complements two related works from the same author. The BEQI methodology characterises per-broker execution quality across five dimensions; the execution-time gap framework (Fesenko, 2026) quantifies how much backtested arbitrage edge survives at a given round-trip time. SBW closes the chain by identifying the time-of-day window over which the strategy operates at peak density. The three works together give the strategy designer a closed-form lens on the question of where, when, and through which broker the strategy can be expected to perform.

Reproducibility

The reference implementation of the SBW algorithm is part of the open-source BJF Arbitrage Statistics WordPress plugin. The synthetic datasets used in the worked examples are documented by parameter in Appendix B; readers can reproduce the datasets by simulation with any standard pseudo-random number generator. The closed-form density score function in equation (1) is elementary and is reproducible in any scientific computing environment.

References

T. G. Andersen and T. Bollerslev. Intraday periodicity and volatility persistence in financial markets. *Journal of Empirical Finance*, 4(2-3):115–158, 1997.

T. Bollerslev and I. Domowitz. Trading patterns and prices in the interbank foreign exchange market. *Journal of Finance*, 48(4):1421–1443, 1993.

M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002)*, pages 635–644, 2002.

R. F. Engle and C. W. J. Granger. Co-integration and error correction: Representation, estimation, and testing. *Econometrica*, 55(2):251–276, 1987.

B. Fesenko. Quantifying the execution-time gap in latency arbitrage backtests: A mathematical framework with empirical validation. *Zenodo*, 2026. doi: 10.5281/zenodo.20616790. URL https://doi.org/10.5281/zenodo.20616790. Preprint.

E. Gatev, W. N. Goetzmann, and K. G. Rouwenhorst. Pairs trading: Performance of a relative-value arbitrage rule. *Review of Financial Studies*, 19(3):797–827, 2006.

J. D. Hamilton. *Time Series Analysis*. Princeton University Press, 1994.

D. A. Hsieh and A. W. Kleidon. Bid-ask spreads in foreign exchange markets: Implications for models of asymmetric information. *The Microstructure of Foreign Exchange Markets*, pages 41–65, 1996.

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, 2003.

J. Kleinberg. Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery*, 7(4):373–397, 2003.

B. N. Lehmann. Fads, martingales, and market efficiency. *Quarterly Journal of Economics*, 105 (1):1–28, 1990.

G. Vidyamurthy. *Pairs Trading: Quantitative Methods and Analysis*. Wiley, 2004.

A Pseudo-code

Algorithm 1: Sliding Best-Window (SBW)

Input: Event stream $E = \{(t_i, m_i)\}_{i=1}^{N}$; bucket size Δ; candidate length set $\mathcal{L}$
Output: Best window $W^* = (b^*_{\text{start}}, L^*)$ and score ρ^*

$B \leftarrow 1440/\Delta$
$\mathbf{n} \leftarrow \text{zeros}(B)$
for each event $(t_i, m_i) \in E$ **do**
    $b \leftarrow \lfloor t_i/\Delta \rfloor$
    $n_b \leftarrow n_b + 1$
end for
$\tilde{\mathbf{n}} \leftarrow [\mathbf{n}, \mathbf{n}]$ *(doubled for modular range queries)*
$\sigma \leftarrow$ prefix sums of $\tilde{\mathbf{n}}$, length $2B + 1$
$\rho^* \leftarrow 0$; $(b^*_{\text{start}}, L^*) \leftarrow (0, 0)$
for each $L \in \mathcal{L}$ **do**
    for $b_{\text{start}} \in \{0, 1, \ldots, B - 1\}$ **do**
        $N \leftarrow \sigma_{b_{\text{start}}+L} - \sigma_{b_{\text{start}}}$
        $s \leftarrow N \cdot \log(1 + N)/(L \cdot \Delta)$
        if s beats ρ^* under tie-breaking rule **then**
            $\rho^* \leftarrow s$; $(b^*_{\text{start}}, L^*) \leftarrow (b_{\text{start}}, L)$
        end if
    end for
end for
return $((b^*_{\text{start}}, L^*), \rho^*)$

Figure 2: Pseudo-code for the Sliding Best-Window algorithm. The tie-breaking rule is: prefer higher s; in case of tie, prefer smaller L; in case of further tie, prefer smaller b_{start}. Complexity: $O(N + B \cdot |\mathcal{L}|)$ time, $O(B)$ space.

B Parameters for Worked Examples

All three worked examples in Section 6 use the following parameters:

- Bucket size $\Delta = 10$ minutes (so $B = 144$)

- Candidate length set in buckets: $\mathcal{L} = \{6, 12, 18, 24, 30, 36, 48, 60, 72, 96, 120\}$, corresponding to wall-clock lengths $\{1\text{h}, 2\text{h}, 3\text{h}, 4\text{h}, 5\text{h}, 6\text{h}, 8\text{h}, 10\text{h}, 12\text{h}, 16\text{h}, 20\text{h}\}$
- Saturation exponent $\alpha = 1.0$ throughout
- Tie-breaking: smallest L, then smallest b_{start} in UTC bucket index

The synthetic event data for each example is generated as follows. For each example, $N = 10{,}000$ events are drawn. Each event's time-of-day is sampled from a mixture distribution defined in the example specification (Section 6): uniform within the dense intervals at the stated probability mass, uniform across the residual hours at the residual probability mass. Each event's metadata field m_i contains a synthetic signed-pip differential drawn from a Pareto distribution with shape $\alpha_{\text{Pareto}} = 2.2$ and scale 0.4 pips; the SBW algorithm does not use the metadata field.