情報処理学会論文誌　データベース　Vol.11　No.2　1–6 (July 2018)

テクニカルノート

ストリーム処理における時系列データの多項式回帰分析手法

今木 常之[1,a)]

受付日 2017年6月9日, 採録日 2017年7月31日

概要：IoT，金融取引，通信などの時系列データに基づくリスク回避，障害予測などを実現するうえで必須となる，近未来予測分析のリアルタイム継続実行技術として，ストリームデータ処理による多項式回帰分析手法を提案する．回帰分析の長期間継続実行においては，時刻値の増大にともなう計算桁溢れを回避するために，継続的な時刻原点の移動と，それにともなう回帰の再計算が必要となる．一方で，再計算のコストはレイテンシスパイク，あるいはスループット低下の要因となる．この課題の解決策として，時刻原点移動のタイミングと移動幅を任意に調整可能な時刻シフト差分計算方式を提案し，移動頻度の削減と計算桁溢れの回避を両立した．本手法を 3 次以上の多項式近似に適用し，回帰再計算のオーバヘッドと計算桁溢れによる近似誤差をともにほぼゼロに抑えることを確認した．このことから，近未来予測分析の無停止実行の実現可能性を示した．

キーワード：ストリームデータ，回帰分析，最小 2 乗法，計算精度

Polynomial Regression Analysis Method of Time Series Data in Stream Processing

Tsuneyuki Imaki[1,a)]

Received: June 9, 2017, Accepted: July 31, 2017

Abstract: We propose a polynomial regression analysis method by stream data processing as a real-time continuous execution technique of near-future forecast analysis, which is essential for realizing risk avoidance, failure prediction, etc. based on time series data such as IoT, financial transactions or telecommunications. In long-period execution of regression analysis, it is necessary to continuously move the time origin and recalculate the regression in order to avoid calculation overflow due to growth of timestamp value. On the other hand, the recalculation cost causes latency spikes or low throughput. As a solution to this problem, we propose an incremental time shift calculation method that can arbitrarily define the timing and length of the time origin movement, and achieved both reduction of movement frequency and avoidance of calculation overflow. We applied this method to third order polynomial approximation and confirmed that both the overhead of the regression recalculation and the approximation error due to calculation overflow are suppressed to almost zero. From this, the feasibility of non-stop execution of near future predictive analysis was shown.

Keywords: stream data, regression analysis, least squares method, calculation precision

1. はじめに

IoT，金融取引，SNS，クリックストリーム，車両の走行データなどの，社会活動で継続的に発生する様々なデータを分析し，ビジネスやサービスの改善にリアルタイムにフィードバックする技術として，ストリームデータ処理（以下，ストリーム処理）があげられる．ストリーム処理の技術分野では，商用のミドルウェア製品群やクラウドサービスのほか，OSS の開発者コミュニティの発展や機械学習との連携も進んでおり，ストリーム処理はビッグデータの一技術分野として定着しつつある．

ストリーム処理による時系列データ分析が効果を持つ活用方法の 1 つが，データの時間推移の予測処理である．金

[1] 株式会社日立製作所
Hitachi Ltd., Kokubunji, Tokyo 185–8601, Japan
[a)] tsuneyuki.imaki.nn@hitachi.com

融取引における価格変動予測，センサ情報に基づく機器故障の予兆検知，渋滞予測など，予測処理はリアルタイム処理の恩恵が最も得られる領域の 1 つである．リアルタイム予測においては，遠い過去のデータによる影響を抑えて，直近のデータに重みをおいて分析する必要がある．この要件に対して，多くのストリーム処理技術は，スライディングウィンドウを備えた継続クエリ言語を提供している．本稿では，SQL をベースとした継続クエリ言語である Continuous Query Language（CQL）[1] に，遅延演算による再帰処理を導入したもの [7] を想定する．

時系列データの観測値から近未来の時間推移を予測する回帰分析の基本的な方法の 1 つとして最小 2 乗法があげられる．特に，分析対象をスライディングウィンドウ上に乗っているデータに絞り込むことで，効果的なリアルタイム予測の実現を期待できる．本稿では，この機能実現において課題となった計算桁溢れの解消方法，およびその評価について報告する．より具体的には次のとおりである．時系列データの近似において，近似曲線の横軸は時間である．一方，ストリーム処理では計算をリアルタイムかつ継続して実行するため，長時間のノンストップ運用では横軸が非常に大きな値となり，近似計算における桁溢れの原因となる．この桁溢れを回避するための最小 2 乗法の変形，および CQL の再帰クエリによるその計算の実現方法を提案する．

2. ストリーム処理による回帰分析の課題

2.1 最小 2 乗法によるウィンドウ上の時系列近似

IoT デバイスなどが生成する時系列データは，データの生成時刻を横軸 x，データの値を縦軸 y とおいて，(x, y) の組の点列が x 軸，すなわち時間軸方向に無限に連続する折れ線グラフととらえることができる．その近未来の変動を予測することは，すでに観測された系列に回帰分析を施して近似式を算出し，その式を時間軸方向に外挿することに相当する．この近似に利用される基本計算が最小 2 乗法である．ここでは，時間 x を独立変数，データ値 y を従属変数として，時刻 x_i における観測値を y_i と表す．また，近似モデルとして高次多項式 $y = a_0 + a_1x + a_2x^2 + \cdots + a_Nx^N$ を想定する．すでに観測されている複数の点 (x_i, y_i) の集合に対して，式 (1) で表される残差の平方和 E を最小化するような係数の組合せを求めることで近似式が得られる．この計算は，式 (2) に示すように各係数による偏微分の値が 0 になる組合せを求めること，すなわち式 (3) に示す連立 1 次方程式の解を求めることに相当する．式 (3) の行列やベクトルの各要素は時刻値 x の冪乗，およびデータ値 y と同冪との積の和であるため，最小 2 乗法の計算は，前半の時刻冪の和の算出と，後半の連立方程式の求解の処理に大きく分かれる．本稿では，以降この前半の計算について考察する．

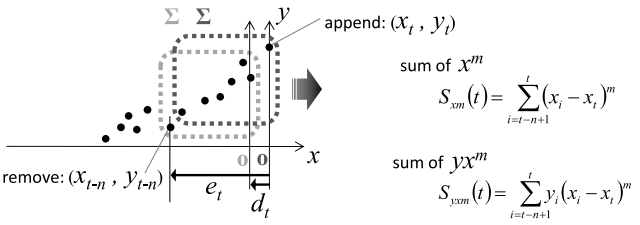

図 1　データの入力にともなう時刻原点の移動と差分計算
Fig. 1　Incremental calculation on shift of the time-series origin at arrival of a new data.

$$E(a_0, a_1, \ldots, a_N) = \sum_i \left(y_i - a_0 - a_1x_i - a_2x_i^2 - \cdots - a_Nx_i^N\right)^2 \quad (1)$$

$$\frac{\partial E}{\partial a_j} = -2\sum_i x_i^j \left(y_i - a_0 - a_1x_i - a_2x_i^2 - \cdots - a_Nx_i^N\right) = 0 \quad (2)$$

$$\begin{pmatrix} \sum x_i^N & \sum x_i^{N+1} & \cdots & \sum x_i^{2N-1} & \sum x_i^{2N} \\ \sum x_i^{N-1} & \sum x_i^N & \cdots & \sum x_i^{2N-2} & \sum x_i^{2N-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \sum x_i & \sum x_i^2 & \cdots & \sum x_i^N & \sum x_i^{N+1} \\ \sum 1 & \sum x_i & \cdots & \sum x_i^{N-1} & \sum x_i^N \end{pmatrix} \times \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \\ a_N \end{pmatrix} = \begin{pmatrix} \sum x_i^N y_i \\ \sum x_i^{N-1} y_i \\ \vdots \\ \sum x_i y_i \\ \sum y_i \end{pmatrix} \quad (3)$$

2.2 時刻原点の移動にともなう計算オーバヘッド

式 (3) の時刻冪の和は，CQL の集約演算である SUM を利用して簡単に定義することが可能である．ただし，係数行列の x は時間軸であるため，x の冪に対する SUM の継続実行は容易に桁溢れにつながる．桁溢れを回避するために倍精度の浮動小数点数（64 bit）を利用した場合は，逆に桁落ちが発生して時刻の精度が失われる．IoT データの近未来予測分析に基づいたサービスを長時間ノンストップで運用するうえで，このような計算精度の劣化は致命的な欠陥となる．この課題の対策として，以下の 3 通りの方法があげられる．

(1) 定期的に時刻原点をシフトし，ウィンドウ上の全データの時刻をシフト後の時刻原点からの相対時刻に変換して，時刻冪の和を再計算
(2) 時刻原点を最新データに追従して移動しつつ（**図 1**）ウィンドウ上のデータ数に依存しない計算量で時刻冪の和を算出するよう，最小 2 乗法の計算式を変形
(3) 時刻を表現するデータの型に，DECIMAL 型（任意精度の十進数）を利用

以前の検討 [8] では (2) を提案方式として，(2)，(3) お

よびSUMを利用する方式を比較評価した（(1)の方式は，レイテンシスパイクが致命的と思われたため対象から除外）．その結果，(3)の方法は，十進数の個々の桁を要素とする整数の配列によって数値を表現するため，大量のCPUリソースとメモリリソースを必要とし，深刻な性能劣化を示すこと，および十進64桁を利用した場合でも，2次以上の式への近似では桁溢れが発生し計算不能エラー（NaN）が発生することを確認した．また，SUMの利用は計算精度の観点で実用に耐えないことを確認した．一方，(2)の方法は，SUMと比較して約半分程度の性能劣化で，4次式への近似でも精度を維持することを確認した．

本報告では，以前の検討においては除外した(1)の方式も含めて再評価する．以降では，(1)の方法をバッチシフト方式，(2)の方法を逐次移動方式と呼ぶ．(1)は時刻原点の移動時に大量の計算が必要であり，瞬間的なレイテンシスパイクを発生させる．一方，(2)はウィンドウサイズに依存しない計算量で個々のデータを処理するためレイテンシスパイクは回避するものの，つねに時刻原点移動が発生するため，(1)と比較して無視できない計算オーバヘッドが発生する．

3. 再帰計算による任意幅の時刻原点移動

3.1 時刻原点移動の差分計算における移動幅の任意設定

2.2節に示したバッチシフト方式と逐次移動方式の2つの時刻原点移動方式は，お互いの利点と欠点が相補的な関係にあった．したがって，両者の利点を取り入れることで欠点を相殺する方法を検討した．具体的には，逐次移動方式において時刻原点の移動頻度を削減するように，バッチシフト方式と同様に，任意のタイミングに任意の幅だけ時刻原点を移動させることが可能となるように，逐次移動方式を改変した．提案方式を次に示す．

時刻tにおけるスライディングウィンドウの範囲をW_tとおくと，時刻値がその範囲に含まれるデータが冪乗和の対象となる．ここで，時刻tにおいて採用している時刻原点の値をxとするとき，ウィンドウ上のデータの時刻値のm乗の和を$S_{xm}(t,x)$，同m乗とデータの積の和を，$S_{yxm}(t,x)$と表す．このとき，時刻tにおいて時刻原点をx_oからx_pに移動することを想定し，移動後の$S_{yxm}(t,x_p)$を式(4)のように変形すると，この値を移動前の$S_{yxk}(t,x_o)$ $(k=0\sim m)$の値から再帰的に算出することが可能となる．$S_{xm}(t,x_p)$についても同様に，式(5)のように変形可能である．

$$
\begin{aligned}
S_{yxm}(t,x_p) &= \sum_{i\in\{i|x_i\in W_t\}} y_i(x_i-x_p)^m \\
&= \sum_i y_i\left\{(x_i-x_o)+x_g\right\}^m \\
&= \sum_{k=0}^{m}\binom{m}{k}x_g^{m-k}\sum_i y_i(x_i-x_o)^k \\
&= \sum_{k=0}^{m}\binom{m}{k}x_g^{m-k}S_{yxk}(t,\,x_o) \qquad (4)
\end{aligned}
$$

$$
S_{xm}(t,x_p)=\sum_{k=0}^{m}\binom{m}{k}x_g^{m-k}S_{xk}(t,\,x_o) \qquad (5)
$$

ただし，$x_g=x_o-x_p$

すなわち，式(6)のような要素b_{ij} $(i,j=1\sim n+1)$を持つ下3角行列$\boldsymbol{B_{n,x_g}}$を定義すると，$S_{yxi}(t,x)$ $(i=0\sim n)$を要素とする$n+1$次元ベクトル$\vec{S}_{yx}(t,x)$について，式(7)のように表すことができる．$S_{xi}(t,x)$ $(i=0\sim n)$を要素とする$n+1$次元ベクトル$\vec{S}_x(t,x)$についても同様に，式(8)のように表すことができる．

$$
b_{ij}=\begin{cases}\binom{i-1}{j-1}x_g^{i-j} & (i\geq j)\\ 0 & (i<j)\end{cases} \qquad (6)
$$

$$
\vec{S}_{yx}(t,x_p)=\boldsymbol{B_{n,x_g}}\vec{S}_{yx}(t,x_o) \qquad (7)
$$

$$
\vec{S}_x(t,x_p)=\boldsymbol{B_{n,x_g}}\vec{S}_x(t,x_o) \qquad (8)
$$

式(7)，(8)より，任意の時刻tにおいて，任意の座標x_pへ時刻原点を移動可能である．ここで，N次多項式への近似に際しては，式(7)において$n=N$，式(8)において$n=2N$とおく．この方式を任意移動方式と呼ぶ．

3.2 任意幅の時刻原点移動を実現する再帰クエリ

3.1節の任意移動方式のクエリは**図 2** に示すデータフ

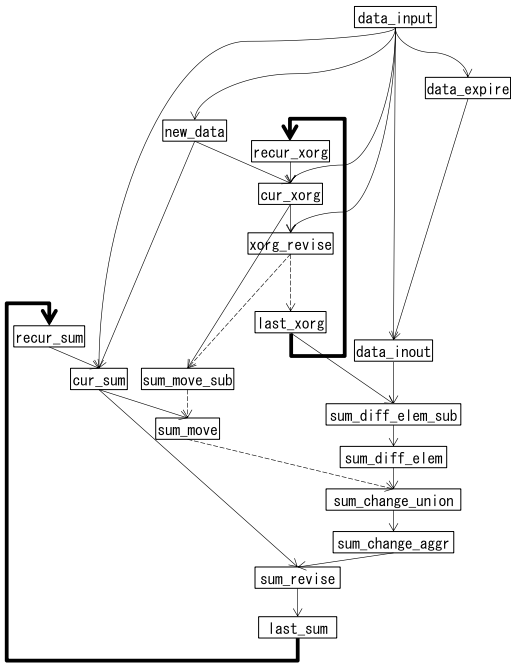

図 2　任意幅で時刻原点移動可能なクエリのデータフロー

Fig. 2　Data flow of the query in which the time-series origin can move arbitrary lengths.

表 1　クエリグラフにおける各クエリの役割
Table 1　Role of the queries in the query graph.

#	クエリ名	役割
1	data_input	入力ストリーム（id: 系列名，x: 時刻値，y: データ値）
2	data_expire	スライディングウィンドウから溢れたデータを出力
3	new_data	新規データ系列の到来を検知
4	recur_xorg	時刻原点の最新値#7を微小時間遅延させて#5にフィードバック(再帰遅延)
5	cur_xorg	時刻原点の現在値を保持
6	xorg_revise	時刻原点の更新値を算出(更新される場合にのみストリームを出力)
7	last_xorg	時刻原点の最新値を保持
8	recur_sum	各和の更新値#18を微小時間遅延させて#9にフィードバック(再帰遅延)
9	cur_sum	各和の現在値を保持
10	data_inout	ウィンドウから出入りしたデータの集合
11	sum_diff_elem_sub	#10の各点について，更新後時刻原点#7からの相対時刻 x を算出
12	sum_diff_elem	相対時刻 x の冪乗，およびデータ値 y との積を算出
13	sum_move_sub	時刻原点が移動した場合に(i.e. #6の出力時に)，移動幅を算出
14	sum_move	式(13),(14)より，時刻原点の移動に伴う各和の変化量を算出
15	sum_change_union	各和の変化量の集合(#12と#14の和集合)
16	sum_change_aggr	#15を集約し，各和の変化量の合計を算出
17	sum_revise	各和の更新値を算出
18	last_sum	各和の最新値を保持

ローで実現され，クエリ#4〜#7 に示される時刻原点更新のループを含むこと以外，逐次移動方式のクエリ構成 [8] と一致する（**表 1** は各クエリの処理内容）．破線で示したパス#6 → #13 → #14 → #15 は，#6 の時刻原点更新の出力がない限り動作しない．式 (7)，(8) の計算は#14 において実行されるが，時刻原点移動の必要がなければその計算は抑制される．

4. 評価

4.1 近似精度の評価

本提案方式の評価として，時刻値の大きさと近似精度との関係を調査した．近似モデルを 1 次式から 4 次式まで変化させ，2.1 節に示した各方式を比較した．各次元について適当なモデル式を定めて生成した系列と，その系列を対象として各方式で求めた近似式との残差の平方和を誤差として評価した．生成系列は 100 点からなる系列の繰り返しとし，誤差の計算は 100 点の処理が終わるごとに実施し，10 回の計算の平均を評価値とした．時刻の値は，生成系列の開始時刻から 1 ずつインクリメントする整数とした．以降，各系列名はそれぞれ以下を表す．

sim.*：CQL の集約演算 SUM をそのまま利用
shift.*：バッチシフト方式
move.*：任意移動方式（逐次移動方式は move において時刻原点の移動幅を 1 にした場合として評価）

図 3 に誤差の評価結果を示す．sim.*では，開始時刻のオーダが増大するにつれて精度が悪化した．浮動小数点（sim.DBL）では計算不可のケースが一部存在した．これは，近似結果として得られた多項式の係数が NaN あるいは無限大などの不正な値となり，誤差計算のために BigDecimal へ変換する際に，実行時例外が発生したことによる．DECIMAL 型（sim.DEC）でも，次数が大きい場合には桁溢れで計算不可のケースが発生した．一方，shift.*, move.*では，時刻の大小にかかわらず，安定した精度を維持した．

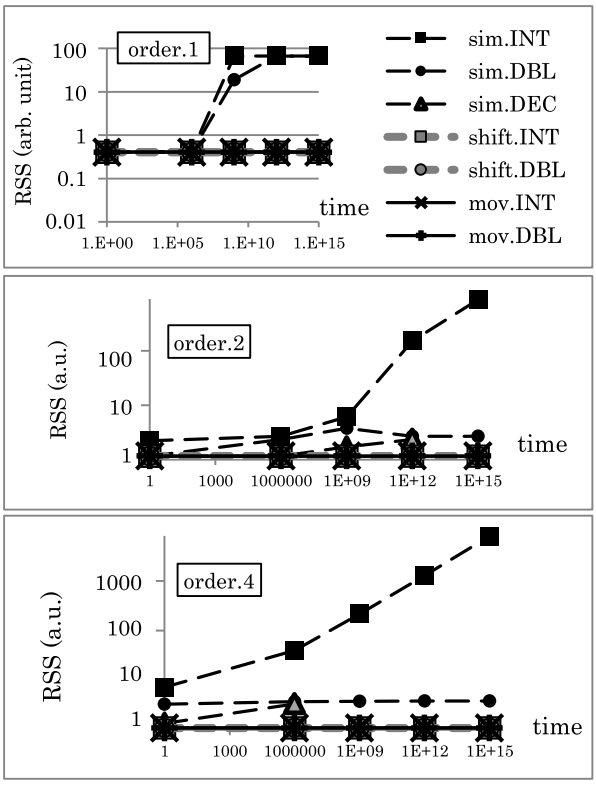

図 3　残差の平方和
Fig. 3　RSS (residual sum of squares).

4.2 時刻原点移動にともなう性能影響の評価

バッチシフト方式と，提案する任意移動方式を対象として，時刻原点の移動にともなう性能影響を評価した．

図 4 は，個々のデータのレイテンシを表す．時刻原点移動幅を 100 として（すなわち，100 データを入力するたびに時刻原点を移動する場合において），20,000 点のデータ系列を処理した際の，末尾のデータのレイテンシ（相対値）を表す．上側のグラフは最後の 1,000 点，下側のグラフはその部分拡大として最後の 300 点のデータを表す．系列名の winXX は，スライディングウィンドウのサイズを表す．バッチシフト方式では，時刻原点移動のタイミングで大きなレイテンシスパイクを確認した．また，ウィンドウサイズの増加にともなうレイテンシスパイクの拡大を確認した．**図 5** は，時刻原点移動タイミングのレイテンシの平均を示す．横軸はウィンドウサイズである．バッチシフト方式では，ウィンドウサイズの増加にともないレイテンシが拡大するのに対し，提案手法ではウィンドウサイズに

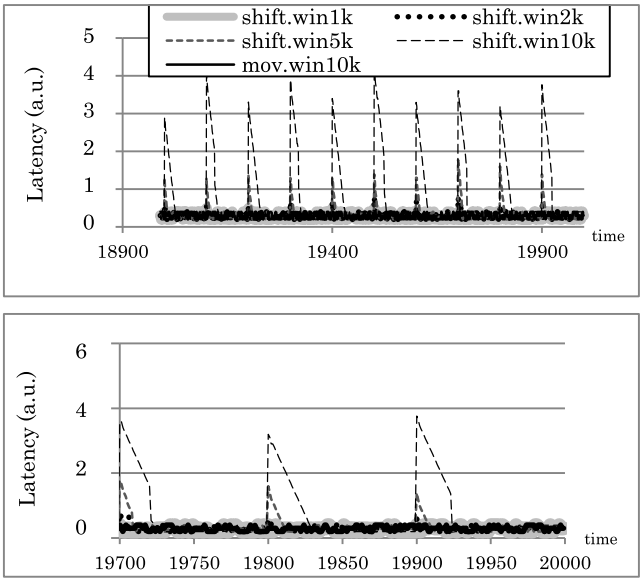

図 4　レイテンシ性能の評価
Fig. 4　Result of latency evaluation.

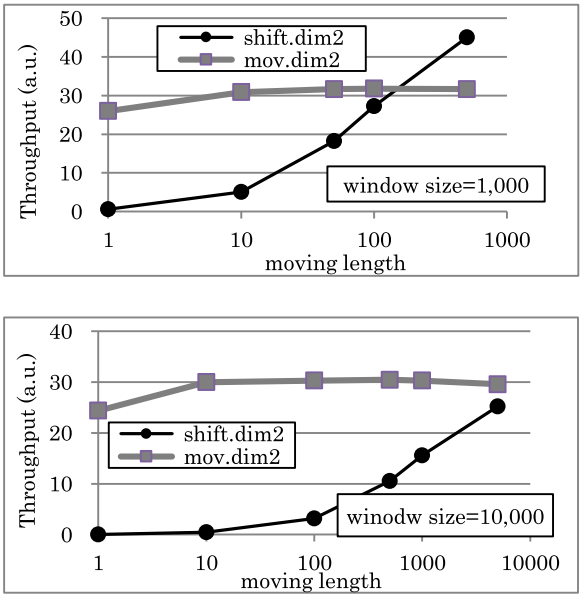

図 6　時刻原点移動幅とスループットの関係
Fig. 6　Throughput on moving length.

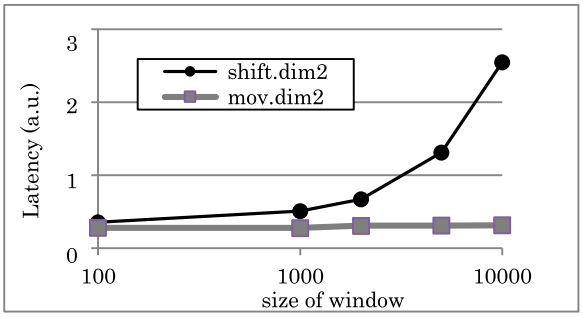

図 5　時刻原点移動時の平均レイテンシ
Fig. 5　Latencies at move of the time-axis origin.

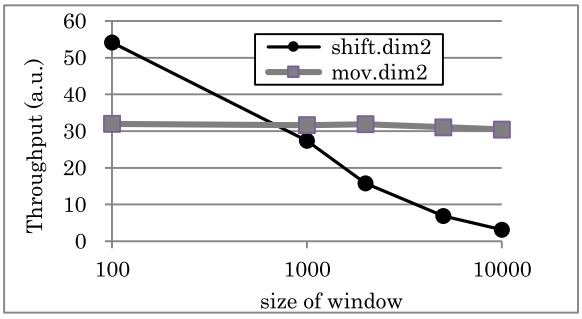

図 7　ウィンドウサイズとスループットの関係
Fig. 7　Throughput on window sizes.

よる性能影響は存在しない．以上，提案方式はストリーム処理において重要な課題であるレイテンシの抑制を，任意のウィンドウサイズで実現可能であることを確認した．

図 6 は，時刻原点移動幅とスループット性能の関係を示す．バッチシフト方式では，移動幅の縮小（すなわち移動頻度の増加）にともなうスループットの劣化が顕著であるのに対し，任意移動方式では移動幅にほぼ依存しないスループットを実現する．実運用では，桁溢れを回避するために，ある程度の頻度で時刻原点移動処理を実行する必要がある．したがって，移動幅に依存しない性能を実現する，提案方式の方が適すると考える．一方，移動幅が 1 の場合は，有意な性能差が確認された．移動幅 1 は逐次移動方式に相当する．すなわち，この結果は，提案方式によって移動タイミングと移動幅を少しでも大きくとれるようにすることで，時刻原点移動の計算オーバヘッドが削減され，性能改善したことを意味している．また，移動幅を 10 以上にしても性能が変化しないことから，提案方式における時刻原点移動計算の性能影響は，移動幅を 10 以上に設定することでほぼゼロにできると考えることができる．

図 7 は，ウィンドウサイズとスループット性能の関係を示す．ウィンドウサイズが小さい場合は，任意移動方式よりもバッチシフト方式の方がシンプルな計算であるため高い性能を示す一方，ウィンドウサイズの増加にともない急激に性能が悪化する．バッチシフト方式では時刻原点移動時の計算コストが多大であるため，移動時に一時的に発生するレイテンシの増大が，全体性能であるスループットにまで影響を及ぼす．一方，提案方式ではウィンドウサイズによらないスループット性能を実現しており，ストリーム処理の元来の特長である，差分処理によるウィンドウサイズ非依存の性能を維持している．

5.　関連研究

最小 2 乗法のインクリメンタルな計算法として，RLS（Recursive Least Squares）アルゴリズムが広く知られている [2]．適応フィルタの古典的手法であり，入力信号を可変フィルタに通した推定値と，望ましい信号値との誤差の 2 乗和を最小化するように，フィルタの係数をインクリメンタルに更新する．係数は過去の入力信号の個定数分に対

して定義され，誤差は忘却係数で重み付けされる．また，RLS の拡張方式として，忘却係数の代わりに，本研究と同様のスライディングウィンドウを利用可能とする方式が，いくつか提案されている [3], [4]．ただし，これら既存研究のいずれにおいても，時間軸の増大にともなう分析精度の劣化の解決を対象とした方式は提案されていない．分析処理を長期間ノンストップで実行するには解決必須の課題であり，その観点では本研究の有効性を主張することが可能と考える．

ストリームデータの再帰処理に関する研究として文献 [5] が関連する．再帰処理中の 2 項演算で punctuation の伝搬が永久にブロックするという課題に対し，閉ループ内に特殊なオペレータ（Flying Fixed-Point）を設け，投機的な punctuation（以下 sp）をループ内で発生させる方式を提案している．ただし，ループ内には同時に 1 つの sp しか流さないことを前提とするため，ループ中に分岐を含むクエリは対象外と推定され，3.2 節に示したデータフローの実行は不可である．これに対し，本稿で利用した方法 [7] にはクエリグラフの構造に関する制約は存在しないため，より広範囲な応用に適用可能である．本研究では，再帰クエリを計算ステートの保持に適用し，その実用性を示した．また，マイクロバッチ型のストリーム処理 [6] と同等であるバッチシフト方式との比較に基づき，データ分析の継続実行におけるレイテンシスパイクを回避することを確認した．

6. まとめ

ストリーム処理における時系列データの近似方法として，その基本方式である最小 2 乗法について，ストリーム処理として実現する際の課題を検討した．その結果，処理の継続による時刻値の増大が，近似計算における桁溢れにつながるという問題を指摘し，その回避策として，時刻原点のスライドをともなう最小 2 乗法の計算方法，および CQL の再帰クエリによる同計算の実現方法を提案した．

さらに，近似精度と計算性能について，ナイーブな計算方法との比較評価を実施し，提案手法が上記の桁溢れを回避すること，および精度と性能を両立する点で，提案手法が優位であることを確認した．

以上の検討結果から，ストリーム処理による時系列データの近似方法，およびその外挿による変動予測の方法として，提案手法が有効であるとの結論を得た．

提案した近似計算方法に基づく予測処理においては，スライディングウィンドウのサイズ，および近似式の外挿期間といったパラメータを，案件ごとにチューニングする必要がある．今後は，パラメータ設定の自動最適化，機械学習，動的設定変更方式などを検討する必要がある．

参考文献

[1] Arasu, A., Babu, S. and Widom, J.: CQL: A Language for Continuous Queries over Streams and Relations, *Proc. DBPL*, pp.1–19 (2003).

[2] Recursive least squares filter, available from ⟨http://en.wikipedia.org/wiki/Recursive_least_squares_filter⟩.

[3] van Vaerenbergh, S., Via, J. and Santamaría, I.: A Sliding-Window Kernel RLS Algorithm and its Application to Nonlinear Channel Identification, *Proc. ICASSP*, pp.789–792 (2006).

[4] Hoagg, J.B., Ali, A.A., Mossberg, M. and Bernstein, D.S.: Sliding Window Recursive Quadratic Optimization with Variable Regularization, *American Control Conference* (*ACC*), pp.3275–3280 (2011).

[5] Chandramouli, B., Goldstein, J. and Maier, D.: On-the-fly Progress Detection in Iterative Stream Queries, *Proc. VLDB*, pp.241–252 (2009).

[6] Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S. and Stoica, I.: Discretized Streams: Fault-Tolerant Streaming Computation at Scale, *Proc. 24th ACM Symposium on Operating Systems Principles*, pp.423–438 (2013).

[7] 今木常之，西澤　格：遅延演算を利用した再帰的ストリームデータ処理の計算完備性，日本データベース学会論文誌，Vol.9, No.3, pp.25–30 (2011).

[8] 今木常之：最小二乗法のストリーム処理における桁あふれ回避方法，第 162 回 DBS 研究発表会，Vol.2015-DBS-162, No.28 (2015).

今木 常之　（正会員）

1996 年東京大学大学院工学系研究科修士課程修了．現在，日立製作所研究開発グループデジタルテクノロジーイノベーションセンタデータサイエンス研究部主任研究員．

（担当編集委員　合田 和生）