Int. J. Appl. Math. Comput. Sci., 2019, Vol. 29, No. 1, 195–206
DOI: 10.2478/amcs-2019-0015

AN ALGORITHM FOR ARBITRARY–ORDER CUMULANT TENSOR CALCULATION IN A SLIDING WINDOW OF DATA STREAMS

KRZYSZTOF DOMINO [a], PIOTR GAWRON [a,*]

[a] Institute of Theoretical and Applied Informatics
Polish Academy of Sciences, Bałtycka 5, 44-100 Gliwice, Poland
e-mail: gawron@iitis.pl

High-order cumulant tensors carry information about statistics of non-normally distributed multivariate data. In this work we present a new efficient algorithm for calculation of cumulants of arbitrary orders in a sliding window for data streams. We show that this algorithm offers substantial speedups of cumulant updates compared with the current solutions. The proposed algorithm can be used for processing on-line high-frequency multivariate data and can find applications, e.g., in on-line signal filtering and classification of data streams. To present an application of this algorithm, we propose an estimator of non-Gaussianity of a data stream based on the norms of high order cumulant tensors. We show how to detect the transition from Gaussian distributed data to non-Gaussian ones in a data stream. In order to achieve high implementation efficiency of operations on super-symmetric tensors, such as cumulant tensors, we employ a block structure to store and calculate only one hyper-pyramid part of such tensors.

Keywords: high order cumulants, time-series statistics, non-normally distributed data, data streaming.

1. Introduction

Cumulants of order one and two of n-dimensional multivariate data, i.e., the mean and the covariance matrix, are widely used in signal and data processing, for example, in one of the most widely used algorithm in data and signal processing, namely, principal component analysis. Cumulants of order one and two describe completely statistical signals or data whose values are governed by a Gaussian distribution. In many real-life cases, data or signals are not normally distributed. In this case it is necessary to employ higher-order cumulants, such as, for example, skewness and kurtosis, to analyze this kind of data.

By the high-order cumulant of n dimensional multivariate data we understand the super-symmetric[1], cumulant tensor $\mathcal{C} \in \mathbb{R}^{[n,d]}$ of $d \geq 3$ modes, each of size n. Importantly they are zeros only if calculated for data sampled from a multivariate Gaussian distribution (Kendall, 1946; Lukacs, 1970). High-order cumulants carry information about the divergence of the empirical distribution from the multivariate Gaussian one. Hence we use them to extract such information from data.

Calculation of higher-order cumulants for multi-dimensional data is time consuming. Furthermore, such data are often recorded in the form of a stream and hence an on-line scheme of calculation and updates of cumulants is useful for their analysis. In this paper we present an efficient algorithm for calculation of cumulants of arbitrary orders in a sliding window for data streams. We show the application of this algorithm to detect a change in the underling distribution of a multivariate time series. Our algorithm uses the so-called block structure, which is a data structure designed for efficient storage and processing of symmetric tensors.

1.1. Motivation. Our motivation to design such an algorithm comes from the fact that there exist many contemporary applications of higher-order cumulant based algorithms in data processing. Typically these employ cumulants up to order four, and rarely up to order six. This limitation comes mainly from two factors: high computational cost of calculating higher-order cumulants and large amounts of data samples required to estimate faithfully higher-order cumulants. Nowadays

*Corresponding author

[1] A tensor is super-symmetric if it is invariant under permutation of its indices.

computational power is widely available and amounts of data collected every day are increasing dramatically. Therefore, we believe that algorithms requiring the usage of high-order cumulants will be employed more widely in the near future. Yet, as pointed out by Stefanowski *et al.* (2017), processing data streams is a challenging task because it imposes constraints on memory usage, processing time, and the number of data input reads. The algorithm presented in this work is dedicated to efficiently process on-line large data streams.

High-order cumulants are used to analyze signal data, such as audio signals, for example, in direction-finding methods of the multi-source signal (q-MUSIC algorithm) (Chevalier *et al.*, 2006). Additionally, high-order cumulants are used in signal filtering problems (Geng *et al.*, 2011; Latimer and Namazi, 2003) or neuroimaging signal analysis (Birot *et al.*, 2011; Becker *et al.*, 2014). The neuroscience application often uses independent components analysis (ICA) (Hyvärinen, 2014), which can be evaluated by means of high-order cumulant tensors (Blaschke and Wiskott, 2004; Virta *et al.*, 2015). Another important issue that requires a fast algorithm to compute and update high-order cumulants is financial data analysis, especially concerning high-frequency financial data, where we deal with large data sets and computational time is a crucial factor. For multi-asset portfolio analysis, high-order cumulant tensors measure risk (Rubinstein *et al.*, 2006; Martin, 2013), especially during a crisis where large fluctuations of asset values are possible (Arismendi Zambrano and Kimura, 2014; Jondeau *et al.*, 2018; Domino, 2017).

While estimating high-order statistics from data, the problem of a high estimation error emerges. In general, large data sets are required for accurate estimation of high-order cumulants from data. This is discussed in detail by Domino *et al.* (2018b). Unfortunately, a large data set requires large computational time, becomes problematic if we want to analyze n-variate data on-line and n is respectively large. To solve this problem, we introduce an algorithm that computes high-order statistics in a sliding window of length t. Statistics are updated every time a new data batch of size t_{up} is collected.

The values of parameters t and t_{up} depend on a particular application. On the one hand, t and t_{up} have to be large enough for an accurate approximation of the statistics; on the other hand, the larger they are, the weaker the time resolution of accessible for application. We typically choose $t_{\mathrm{up}} = \alpha t$ with $\alpha = 2.5\%, 5\%, \ldots$ Given such parameters, we have reached over an order of magnitude speedup compared with a simple cumulant recalculation using the fast algorithm introduced by Domino *et al.* (2018b). In both cases we use the block structure (Schatz *et al.*, 2014) that allows calculating and storing efficiently super-symmetric cumulant and moment tensors (Domino *et al.*, 2018b). We show that using the presented algorithm we can analyze data recorded at frequencies up to 2000 Hz from 150 Hz, depending on the number of marginal variables n: 60—for the higher frequency figure, to 120—for the lower figure, on a modern six-core workstation.

1.2. Paper structure. The paper is organized as follows. In Section 2 we present formulas and the algorithm employed to calculate cumulants of a data stream, the input data format, the sliding window mechanism, the block structure, moment tensor updates, cumulant calculation, and complexity analysis. In Section 3 we discuss an algorithm implementation in the Julia programming language and performance tests of the implementation. In Section 4 we introduce an illustrative application of our algorithm employed to analyze the on-line statistics of a data stream and the maximal frequency of data that can be calculated on-line given specific computer hardware.

2. Statistics updates

2.1. Data format. Consider data that consist of t realizations sampled from an n-dimensional multivariate distribution forming an observation window whose number will be indexed by w:

$$\mathbb{R}^{t\times n} \ni \mathbf{X}^{(w)} = \begin{bmatrix} x_{1,1}^{(w)} & \cdots & x_{1,n}^{(w)} \\ \vdots & \ddots & \vdots \\ x_{t,1}^{(w)} & \cdots & x_{t,n}^{(w)} \end{bmatrix}. \tag{1}$$

Note that samples form rows in the data matrix.

Further consider an update which consists of other t_{up} n-dimensional realizations

$$\mathbb{R}^{t_{\mathrm{up}}\times n} \ni \mathbf{X}_{(+)}^{(w)} = \begin{bmatrix} x_{t+1,1}^{(w)} & \cdots & x_{t+1,n}^{(w)} \\ \vdots & \ddots & \vdots \\ x_{t+t_{\mathrm{up}},1}^{(w)} & \cdots & x_{t+t_{\mathrm{up}},n}^{(w)} \end{bmatrix}, \tag{2}$$

which will be concatenated into $\mathbf{X}^{(w)}$ in order to form a new window. Additionally the forming of a new window will require dropping the first t_{up} realizations represented by the following matrix:

$$\mathbb{R}^{t_{\mathrm{up}}\times n} \ni \mathbf{X}_{(-)}^{(w)} = \begin{bmatrix} x_{1,1}^{(w)} & \cdots & x_{1,n}^{(w)} \\ \vdots & \ddots & \vdots \\ x_{t_{\mathrm{up}},1}^{(w)} & \cdots & x_{t_{\mathrm{up}},n}^{(w)} \end{bmatrix}. \tag{3}$$

The new observation window $w+1$ is given by the

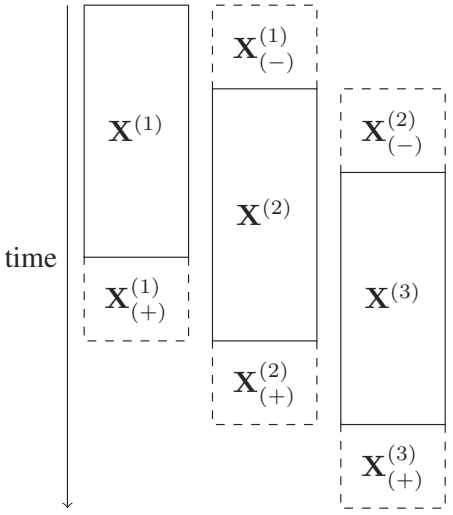


Fig. 1. Schematic representation of a data flow in the sliding window mechanism. In the picture the time flows from top to bottom. Subsequent windows are placed from left to right. Each multivariate data sample forms a row of a matrix.

following equation:

$$\begin{aligned}\mathbb{R}^{t\times n} \ni \mathbf{X}^{(w+1)} &= \begin{bmatrix} x^{(w)}_{t_{\mathrm{up}}+1,1} & \dots & x^{(w)}_{t_{\mathrm{up}}+1,n} \\ \vdots & \ddots & \vdots \\ x^{(w)}_{t+t_{\mathrm{up}},1} & \dots & x^{(w)}_{t+t_{\mathrm{up}},n} \end{bmatrix} \\ &= \begin{bmatrix} x^{(w+1)}_{1,1} & \dots & x^{(w+1)}_{1,n} \\ \vdots & \ddots & \vdots \\ x^{(w+1)}_{t,1} & \dots & x^{(w+1)}_{t,n} \end{bmatrix}.\end{aligned} \tag{4}$$

The sliding window mechanism is visualized in Fig. 1.

2.2. Sliding window. The algorithm presented in this work calculates cumulants of a data stream in a sliding window. It is assumed that data arrive continuously and are fed to the algorithm in typically small batches. The algorithm uses only a subset of current data stored in a buffer and minimal required statistics. As new data are incoming, the calculations are performed on stored data and statistics. Historical data are iteratively discarded. The main loop is summarized in Algorithm 1, which consists of the following steps: acquire new batch of data; calculate the oldest batch of data; update moments; calculate cumulants; update the data buffer.

2.3. Block structure. Moments and cumulants are super-symmetric tensors. Therefore we use a block structure as introduced by Schatz *et al.* (2014) to compute and store them effectively. Using such a block structure, we store and compute only one hyper-pyramidal part of the super-symmetric tensor in blocks of size b^d, where b is a parameter of the storage method. One advantage of the block structure is that it allows further efficient processing of cumulants, which was discussed by Domino *et al.* (2018b).

2.4. Moment tensor updates. Given data $\mathbf{X} \in \mathbb{R}^{t\times n}$, the super-symmetric moment tensor of order d: $\mathcal{M}_d(\mathbf{X}) \in \mathbb{R}^{[n,d]}$ consists of the following elements:

$$m_{\mathbf{i}}(\mathbf{X}) = \frac{1}{t}\sum_{l=1}^{t}\left(\prod_{i_k\in\mathbf{i}} x_{l,i_k}\right), \tag{5}$$

where $\mathbf{i} = (i_1,\dots,i_d)$ is the element's multi-index and $i_1,\dots,i_d \in 1:n$. A naïve approach to calculate moments $\mathcal{M}\left(\mathbf{X}^{(w)}\right)$ would be to calculate all $m_{\mathbf{i}}(\mathbf{X}^{(w)})$ for each window w. But in order to reduce the amount of computation required to calculate moments in sliding windows, we take advantage of the fact that, given $\mathbf{X}^{(w)}_{(-)}$ and $\mathbf{X}^{(w)}_{(+)}$, it is easy to update each element of the moment tensor using the following relation:

$$\begin{aligned} & m_{\mathbf{i}}\left(\mathbf{X}^{(w+1)}\right) \\ &= \frac{1}{t}\sum_{l=1+t_{\mathrm{up}}}^{t+t_{\mathrm{up}}}\left(\prod_{i_k\in\mathbf{i}} x^{(w)}_{l,i_k}\right) \\ &= \frac{1}{t}\sum_{l=1}^{t}\left(\prod_{i_k\in\mathbf{i}} x^{(w)}_{l,i_k}\right) + \frac{t_{\mathrm{up}}}{t}\left(\frac{1}{t_{\mathrm{up}}}\sum_{l=1+t}^{t+t_{\mathrm{up}}}\left(\prod_{i_k\in\mathbf{i}} x^{(w)}_{l,i_k}\right)\right. \\ &\quad \left. -\frac{1}{t_{\mathrm{up}}}\sum_{l=1}^{t_{\mathrm{up}}}\left(\prod_{i_k\in\mathbf{i}} x^{(w)}_{l,i_k}\right)\right) \\ &= m_{\mathbf{i}}\left(\mathbf{X}^{(w)}\right) + \frac{t_{\mathrm{up}}}{t}\left(m_{\mathbf{i}}\left(\mathbf{X}^{(w)}_{(+)}\right) - m_{\mathbf{i}}\left(\mathbf{X}^{(w)}_{(-)}\right)\right). \end{aligned} \tag{6}$$

We can write Eqn. (6) using tensor notation in the following tensor form:

$$\begin{aligned}\mathcal{M}(\mathbf{X}^{(w+1)}) &= \mathcal{M}(\mathbf{X}^{(w)}) \\ &\quad + \frac{t_{\mathrm{up}}}{t}(\mathcal{M}(\mathbf{X}^{(w)}_{(+)}) - \mathcal{M}(\mathbf{X}^{(w)}_{(-)})).\end{aligned} \tag{7}$$

Exploiting this form, we can write Algorithm 2, which calculates moments in a sliding window $w+1$ given moments of window w, and the data batches $\mathbf{X}^{(w)}_{(-)}$ and $\mathbf{X}^{(w)}_{(+)}$.

There exists a different approach to this problem. We could calculate t/t_{up} moments of data batches and organize the moments in a FIFO queue,

$$\left(\mathcal{M}(\mathbf{X}^{(w_1)}_{(+)}), \mathcal{M}(\mathbf{X}^{(w_2)}_{(+)}), \dots, \mathcal{M}(\mathbf{X}^{(w_{t/t_{\mathrm{up}}})}_{(+)})\right). \tag{8}$$

Algorithm 1. Calculation of sliding window cumulants.

Require: $\mathbf{X}^{(1)}$: first data batch
Ensure: $\mathcal{C}_1(\mathbf{X}^{(w)}), \ldots, \mathcal{C}_d(\mathbf{X}^{(w)})$: cumulants for windows $1:w$
1: Calculate moments $\mathcal{M}_1(\mathbf{X}^{(1)}), \ldots, \mathcal{M}_d(\mathbf{X}^{(1)})$
2: $w \leftarrow 1$
3: **for** w **do**
4: Acquire $\mathbf{X}^{(w)}_{(+)}$.
5: Calculate $\mathbf{X}^{(w)}_{(-)}$ from first t_{up} rows of $\mathbf{X}^{(w)}$.
6: $\mathcal{M}_1(\mathbf{X}^{(w+1)}), \ldots, \mathcal{M}_d(\mathbf{X}^{(w+1)}) \leftarrow$ momentsupdate$(\mathcal{M}_1(\mathbf{X}^{(w)}), \ldots, \mathcal{M}_d(\mathbf{X}^{(w)}), \mathbf{X}^{(w)}_{(+)}, \mathbf{X}^{(w)}_{(-)})$
7: $\mathcal{C}_1(\mathbf{X}^{(w+1)}), \ldots, \mathcal{C}_d(\mathbf{X}^{(w+1)}) \leftarrow$ mom2cums$(\mathcal{M}_1(\mathbf{X}^{(w+1)}), \ldots, \mathcal{M}_d(\mathbf{X}^{(w+1)}))$
8: Calculate $\mathbf{X}^{(w+1)}$ by concatenating row by row $\mathbf{X}^{(w)}$ with $\mathbf{X}^{(w)}_{(+)}$ and remove rows belonging to $\mathbf{X}^{(w)}_{(-)}$.
9: Emit $\mathcal{C}_1(\mathbf{X}^{(w+1)}), \ldots, \mathcal{C}_d(\mathbf{X}^{(w+1)})$.
10: $w \leftarrow w + 1$
11: **end for**

Algorithm 2. `momentsupdate()`.

Require: data: $\mathbf{X}^{(w)}_{(+)} \in \mathbb{R}^{t_{up} \times n}$, $\mathbf{X}^{(w)}_{(-)} \in \mathbb{R}^{t_{up} \times n}$, moments: $\mathcal{M}_1(\mathbf{X}^{(w)}), \ldots, \mathcal{M}_d(\mathbf{X}^{(w)})$.
Ensure: updated moments: $\mathcal{M}_1(\mathbf{X}^{(w+1)}), \ldots, \mathcal{M}_d(\mathbf{X}^{(w+1)})$
1: **for** $s \leftarrow 1$ **to** d **do**
2: $\mathcal{M}_s(\mathbf{X}^{(w+1)}) \leftarrow \mathcal{M}_s(\mathbf{X}^{(w)}) + \frac{t_{\mathrm{up}}}{t}\left(\mathcal{M}\big(\mathbf{X}^{(w)}_{(+)}\big) - \mathcal{M}\big(\mathbf{X}^{(w)}_{(-)}\big)\right)$
3: **end for**{see Eqn. (7)}
4: **return** $\mathcal{M}_1(\mathbf{X}^{(w+1)}), \ldots, \mathcal{M}_d(\mathbf{X}^{(w+1)})$

With the arrival of new batch its moments would be calculated and added to the aggregate moments; then the oldest batch moments would be subtracted from the aggregate. This scheme reduces the amount of calculations because it does not require to calculate moments of $\mathbf{X}^{(w)}_{(-)}$ for each window w, but requires the storage of t/t_{up} moments for data $\mathbf{X}^{(w_b)}_{(+)}$ for $w_b \in w : w + t/t_{\mathrm{up}}$. Therefore, in our approach we propose to trade some of the computational complexity for a reduction in memory size requirements.

Moment tensor computation and storage in the block structure are explained in detail by Domino *et al.* (2018b), thanks whom we can conclude that if $b \ll n$ and $d \ll n$ we need approximately $(n^d/d!)(d-1)t$ multiplications to compute $\mathcal{M}_d(\mathbf{X})$. Analogously, we need $(n^d/d!)(d-1)t_{\mathrm{up}}$ multiplications to compute $\mathcal{M}\big(\mathbf{X}^{(w)}_{(+)}\big)$ and the same number of multiplications to compute $\mathcal{M}\big(\mathbf{X}^{(w)}_{(-)}\big)$. Obviously, simple recalculation of $\mathcal{M}(\mathbf{X}^{(w+1)})$ would require $(n^d/d!)(d-1)t$ multiplications. Hence, given $\mathcal{M}\left(\mathbf{X}^{(w)}\right)$ computed beforehand, the theoretical speedup factor of the update compared with simple recalculation of $\mathcal{M}\left(\mathbf{X}^{(w+1)}\right)$ would be

$$\frac{\frac{n^d}{d!}(d-1)t}{\frac{n^d}{d!}(d-1)t_{\mathrm{up}} + \frac{n^d}{d!}(d-1)t_{\mathrm{up}}} = \frac{t}{2t_{\mathrm{up}}}, \qquad (9)$$

which is significant especially if $t_{\mathrm{up}} \ll t$. In the next two subsections we are going to show how to use the moment update scheme to update cumulant tensors.

2.5. Cumulant updates calculation. Given the moment tensor update scheme, due to the recursive relation between moments and cumulant tensors (Barndorff-Nielsen and Cox, 1989), we can use this scheme to form cumulants' update algorithm. The recursive relation between cumulants and moments was discussed in detail by Domino *et al.* (2018b). Here this relation is summarized in the form of Algorithm 3. It calculates cumulants' tensors $\mathcal{C}_s(\mathbf{X}) \in \mathbb{R}^{[n,s]}$ for orders $s \in \{1, 2, \ldots, d\}$, given moments $\mathcal{M}_1(\mathbf{X}), \ldots, \mathcal{M}_d(\mathbf{X})$.

2.6. Complexity analysis. Despite using the cumulant-moment recursive relation (Domino *et al.*, 2018b), there are important computational differences between the cumulants' updated scheme proposed in this paper and the calculation scheme proposed by Domino *et al.* (2018b); see Eqn. (34). In the first case,

(i) we need much fewer arithmetic operations to update a moment tensor than in the second case since $t_{\mathrm{up}} \ll t$, but

(ii) we need slightly more computational power to compute $\mathcal{A}$ in Algorithm 4. In the first case

Algorithm 3. `moms2cums()`.

Require: $\mathcal{M}_1(\mathbf{X}), \ldots, \mathcal{M}_d(\mathbf{X})$: moments
Ensure: $\mathcal{C}_1(\mathbf{X}), \ldots, \mathcal{C}_d(\mathbf{X})$: cumulants
1: **for** $s \leftarrow 1$ **to** d **do**
2: $\mathcal{C}_s(\mathbf{X}) \leftarrow \mathcal{M}_s(\mathbf{X}) : \mathcal{A}$ {Calculate elements of $\mathcal{A}$ using Algorithm 4}
3: **end for**
4: **return** $\mathcal{C}_1(\mathbf{X}), \ldots, \mathcal{C}_d(\mathbf{X})$

Algorithm 4. Calculation of a symmetrized outer product.

Require: $\mathbf{i}$: multi-index of cumulant tensor, s: order of the cumulant being calculated, $\mathcal{C}_1(\mathbf{X})$, $\mathcal{C}_2(\mathbf{X})$, ..., $\mathcal{C}_{s-1}(\mathbf{X})$: cumulant tensors of lower orders
Ensure: $\mathcal{A}_\mathbf{i}$: element of super-symmetric tensor $\mathcal{A}$
1: $\mathcal{A}_\mathbf{i} = 0$
2: **for** $\sigma \leftarrow 2$ **to** s **do**
3: calculate partitions of the set $1 : s$ into σ parts {using Knuth's algorithm (Knuth, 2011, Section 7.2.1.4)}
4: **for** $\xi \in$ partitions **do**
5: $\mathrm{a} \leftarrow 1$
6: **for** $k \in \xi$ **do**
7: $\mathrm{a} \leftarrow \mathrm{a} \times \mathcal{C}_{\mathbf{i}(k)}(\mathbf{X})$
8: **end for**
9: $\mathcal{A}_\mathbf{i} \leftarrow \mathcal{A}_\mathbf{i} + \mathrm{a}$
10: **end for**
11: **end for**

we cannot use central moments for $\mathcal{M}_d$ because, in general, updates affect the centering of the data. Hence in the first case the inner loop starting in Line 4 of Algorithm 4 runs over all partitions, contrary to the second case, where a similar algorithm sums over partitions containing only elements of size ≥ 2.

In order to analyze the computational complexity of the sliding window cumulant calculation algorithm, we have to count the number of multiplications performed in Line 7 of Algorithm 4. This number is given by

$$\begin{aligned}\sum_{\sigma=1}^{d} S(d,\sigma)(\sigma-1) &= \sum_{\sigma=2}^{d} S(d,\sigma)(\sigma-1) \\ &\leq (d-1)\sum_{\sigma=2}^{d} S(d,\sigma) \\ &< (d-1)B(d),\end{aligned} \tag{10}$$

where $S(d,\sigma) > 0$ is the number of partitions of a set of size d into σ parts, i.e., the Stirling number of the second kind (Graham *et al.*, 1989); the sum $\sum_{\sigma=1}^{d} S(d,\sigma) = B(d)$ is the Bell number (Comtet, 1974), the number of all partitions of the set of size d. The upper limit $(d-1)B(d)$ will be used further to approximate the number of multiplications required.

The number of multiplications is reduced due to the use of the block storage of super-symmetric tensors. We need only to calculate approximately $n^d/d!$ tensor elements (Domino *et al.*, 2018b). Given a moment tensor $\mathcal{M}_d$ and cumulant tensors $\mathcal{C}_1, \ldots, \mathcal{C}_{d-1}$ we can approximate the number of multiplications to compute $\mathcal{C}_d$ by

$$\frac{n^d}{d!}(d-1)B(d). \tag{11}$$

Nevertheless, it is important to note that there is some additional computational overhead in the implementation due to operations on relatively small blocks.

Referring to Eqn. (9) in order to update a series of moments, we need approximately

$$\#N_{\mathrm{mup}}(d) \approx \sum_{k=1}^{d} 2\frac{n^k}{k!}(k-1)t_{\mathrm{up}} \tag{12}$$

multiplications. Further, according to Eqn. (11), to compute a series of cumulant tensors, given a series of moment tensors, we need approximately

$$\sum_{k=1}^{d} \frac{n^k}{k!}(k-1)B(k) \tag{13}$$

multiplications. Finally, to update a series of cumulants, we need

$$\begin{aligned}\#N_{\mathrm{cup}} &\approx \sum_{k=1}^{d} 2\frac{n^k}{k!}(k-1)t_{\mathrm{up}} + \sum_{k=1}^{d} \frac{n^k}{k!}(k-1)B(k) \\ &= \sum_{k=1}^{d} 2\frac{n^k}{k!}(k-1)(2t_{\mathrm{up}} + B(k))\end{aligned} \tag{14}$$

multiplications. In practice, we use cumulant orders of $d = 4, 5, 6$, the number of data $t > 10^5$, the batch size of $t_{\mathrm{up}} = \alpha t$, where $\alpha = 2.5\%, 5\%, \ldots$, and the number of variables $n \gg d$. Further, given that the Bell number $B(d)$ grows rapidly with d (Comtet, 1974), the last term of the sum in Eqn. (14) is dominant and hence the final number of multiplications can be approximated by

$$\#N_{cup} \approx \frac{n^d}{d!}(d-1)(2t_{\mathrm{up}} + B(d)). \tag{15}$$

Simple cumulant series recalculation using the results Domino *et al.* (2018b) requires approximately

$$\sum_{k=1}^{d} \frac{n^k}{k!}(k-1)t \approx \frac{n^d}{d!}(d-1)t$$

multiplications. The final speedup factor compared with such recalculation is

$$\frac{t}{2t_{\mathrm{up}} + B(d)}. \tag{16}$$

The first term in the denominator corresponds to Algorithm 2, while the second to Algorithm 3. For the analyzed parameter values, the function `momentsupdate()` is more computationally costly in comparison with `moms2cums()` by a factor of $2t_{\mathrm{up}}/B(d)$. For example, for $t_{\mathrm{up}} = 25000$ and $d = 4$, this factor is of three orders of magnitude.

In the following sections we present computer implementation of the cumulant updates algorithm in the `Julia` programming language, along with performance tests.

3. Implementation and performance

3.1. Implementation. The sliding window cumulant calculation algorithm was implemented in the `Julia` programming language (Bezanson *et al.*, 2012; 2017; 2014) and provided in the Zenodo repository (Domino and Gawron, 2018). Our implementation uses the block structure given by Domino *et al.* (2017) and parallel computation via the function `pmap()` implemented in the `Julia` programming language. For parallel computation implementation we perform the following steps.

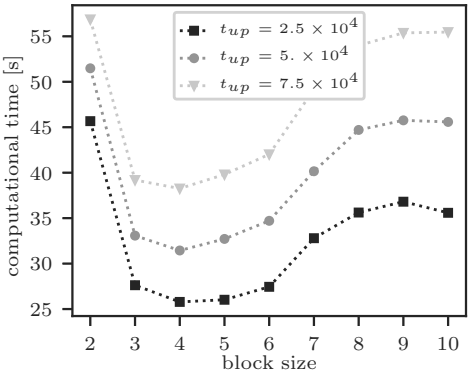
1. We use parallel implementation of moment tensor calculation introduced by Domino *et al.* (2018b), i.e., data are split into p non-overlapping sub-series, where p is the number of workers; next, we compute moment tensors for each sub-series and combine them into a single moment tensor.

2. We have also parallelized the 'for' loop in line 2 of Algorithm 4 using the `pmap()` function, which is one of the ways `Julia` programming language implements a parallel for. The advantage of this solution is that each term of that sum is super-symmetric and we can compute it using a block structure. The disadvantage is that the sum has only $d - 1$ elements. Hence for a large number of workers we do not take full advantage of parallel implementation.

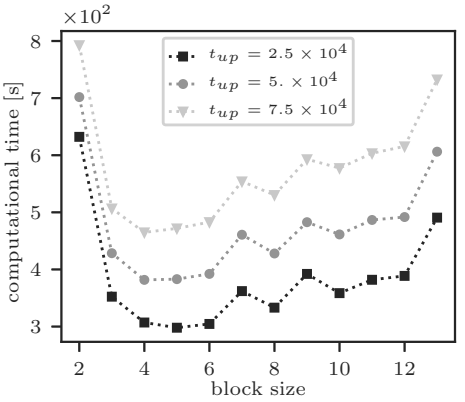
Despite some inefficiencies of parallel implementation, we obtain large speedup due to multiprocessing, which is presented below.

3.2. Performance tests. In what follows we present performance tests carried out using mainly multiple CPU cores. All tests were performed on a computer equipped with an Intel(R) Core(TM) i7-6800K CPU @ 3.40 GHz processor, providing 6 physical cores and 12 computing cores with hyper-threading, and 64 GB of random access memory.

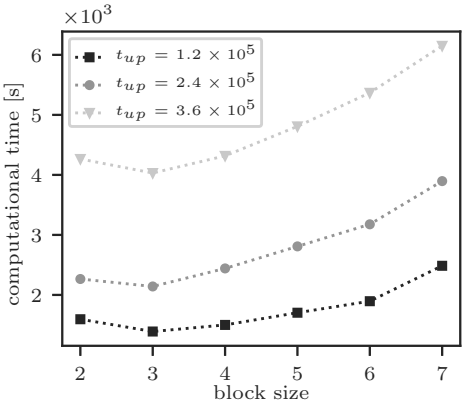
We start with determining the optimal block size parameter b of the block structure (see Section 2.3). This parameter has profound impact on the computational time.



(a) $d = 4, n = 60$



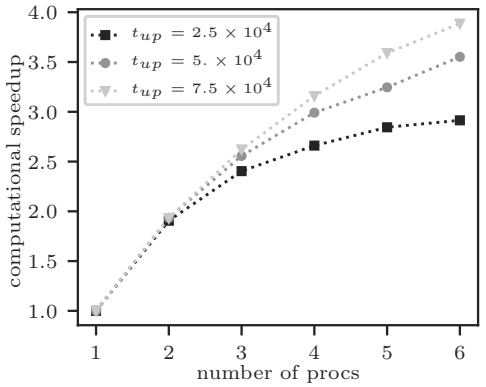
(b) $d = 4, n = 120$



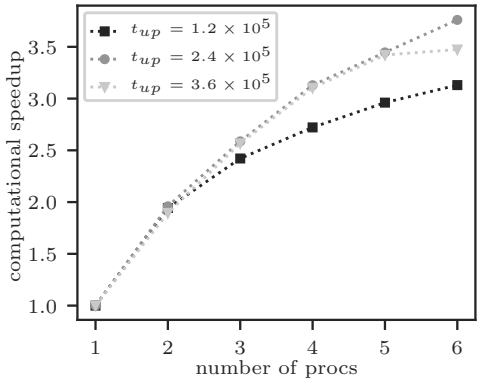
(c) $d = 5, n = 60$

Fig. 2. Computational times of cumulant updates for different block sizes b and multiprocessing implementation on 6 workers.

On the one hand, the higher b, the more computational and storage redundancy while calculating moment tensors, due to larger diagonal blocks. On the other hand, the lower b, the more computational overhead due to a larger amount of operations performed on small blocks.



(a) $d = 4, n = 120, b = 4$



(b) $d = 5, n = 60, b = 3$

Fig. 3. Performance tests for multiprocessing implementation.

In Fig. 2 we present the computational time of the update of cumulant tensors series of order $1, \ldots, d$ for different block sizes. One can observe that the higher cumulant order d, the lower optimal block size b.

Finally, fluctuations in computational time vs. the block size are caused by the fact that, in our implementation, if b does not divide n, some blocks are not hyper-squares and hence calculation of their size and block size conversion cause additional computational overhead. The computations of the optimal block size were performed using 6 parallel worker processes.

Scalability of the algorithm with an increase in the number of CPU cores is presented in Fig. 3. At first the computational time speedup is proportional to the number of workers as should be expected; however, for a large number of workers we do not fully take advantage of the parallel implementation which is discussed in a previous section. Despite this problem we still have a large speedup due to use of multiple cores.

In Fig. 4 we present the computational speedup of the update cumulants of order $1, \ldots, d$, compared with their simple recalculation (Domino, 2017) implemented in `Julia` (Domino *et al.*, 2018a). The main conclusion is that the computational speedup is of about one order of magnitude. Higher speedup is recorded for large data sets.

4. Illustrative application

In this section we show a practical application of the sliding window cumulant calculation algorithm to analyze data that are updated in batches. As a simple application we propose the following scenario. The initial batch of data $\mathbf{X}^{(1)}$ is sampled from a multivariate Gauss distribution. Then the subsequent update batches $\mathbf{X}^{(w)}_{(+)}$ are drawn from the t-Student copula—a strongly non-Gaussian distribution—having the same univariate marginal as the Gauss distribution. The transition from a Gaussian to a non-Gaussian regime is observed using the value of the Frobenius norm of the fourth cumulant tensor.

4.1. Cumulant based measures of data statistics.

According to the definition of high-order cumulants (Kendall, 1946; Lukacs, 1970), they are zero only if data are sampled from a multivariate Gaussian distribution. Hence in this case the Frobenius norm of a high order cumulant tensor,

$$\|\mathcal{C}_d\|_k = \sqrt[k]{\sum_{\mathbf{i}} |c_{\mathbf{i}}|^k}, \tag{17}$$

should be zero as well. Introduce the function

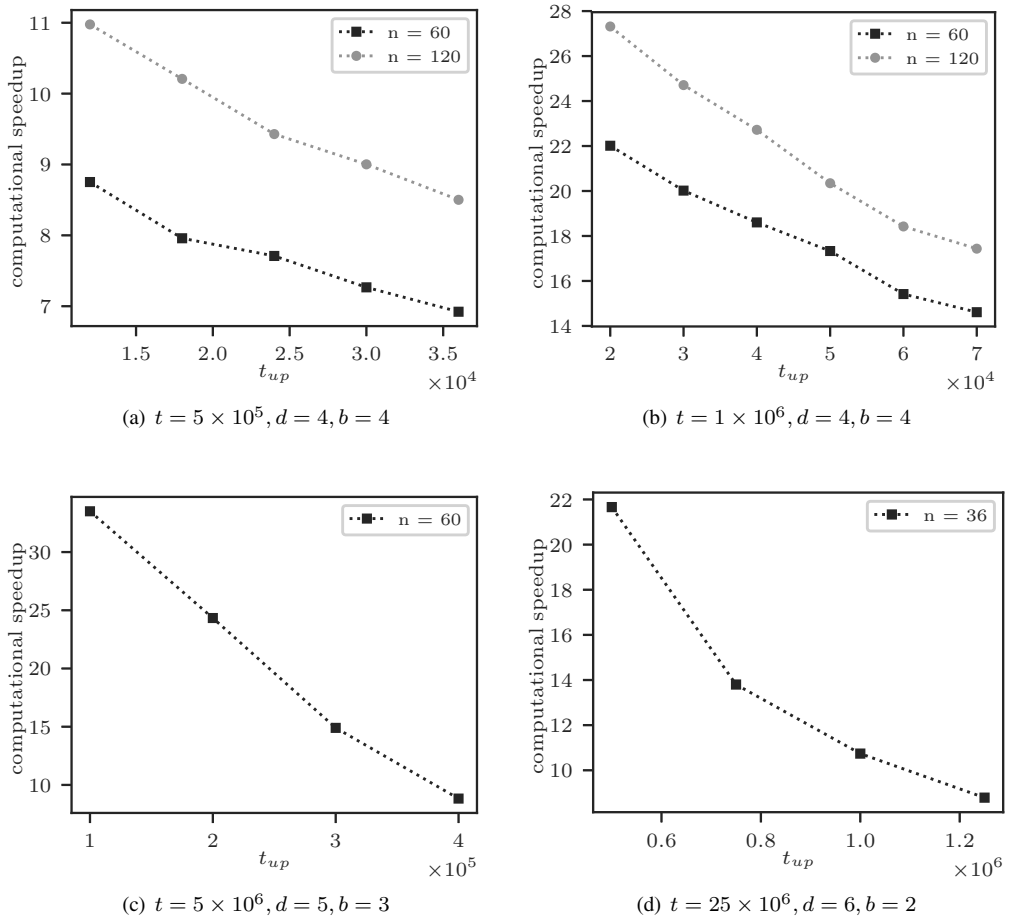
$$\nu_d = \frac{\|\mathcal{C}_d\|_2}{\|\mathcal{C}_2\|_2^{d/2}} \quad \text{for } d > 2, \tag{18}$$

which will be used to detect non-Gaussianity of data. Recall that, in the case of a univariate random variable, for $d = 3$ and $d = 4$ the function ν_d is equal to the modules of asymmetry and kurtosis, respectively. Obviously, for multivariate data, the higher values of ν_d, the less likely that data were sampled from a multivariate Gaussian distribution.

Due to the use of the block structure (Schatz *et al.*, 2014; Domino *et al.*, 2018b), the function ν_d can be computed fast and use a small amount of memory

Suppose we have a supper-symmetric cumulant tensor $\mathcal{A} \in \mathbb{R}^{[n,d]}$ stored in a block structure, i.e., we store only one hyper-pyramidal part of such a tensor in blocks. Let $\mathbf{j} = (j_1, \ldots, j_d)$ be a multi-index of block $(\mathcal{A})_{\mathbf{j}} \in \mathbb{R}^{b^d}$; with no loss of generality and for the sake of simplicity, we assume that $b|n$. Then, in the block structure, we store only blocks indexed by such $\mathbf{j}$ whose elements are sorted in a increasing order.

We propose Algorithm 5, which computes a k-norm of given super-symmetric tensor $\mathcal{A} \in \mathbb{R}^{[n,d]}$. Blocks in a block structure can be super-diagonal (super-symmetric), partially diagonal (partially-symmetric) or off-diagonal.



(a) $t = 5 \times 10^5, d = 4, b = 4$ (b) $t = 1 \times 10^6, d = 4, b = 4$

(c) $t = 5 \times 10^6, d = 5, b = 3$ (d) $t = 25 \times 10^6, d = 6, b = 2$

Fig. 4. Speedup of cumulant updates compared with the results by Domino *et al.* (2018b): a six-worker implementation.

Algorithm 5. Calculation of the k-norm of the tensor stored in a block structure.

Require: $\mathcal{A} \in \mathbb{R}^{[n,d]}$: the supper-symmetric tensor stored in blocks, $\bar{n}$ – number of blocks

Ensure: Number: the k-norm of the tensor

1: $z \leftarrow 0$
2: **for** $j_1 \leftarrow 1$ **to** $\bar{n}, \ldots, j_d \leftarrow j_{d-1}$ **to** $\bar{n}$ **do**
3: $\mathbf{j} = (j_1, \ldots, j_d)$
4:

$$z \leftarrow z + \frac{d!}{\prod_l r_l!} \sum_{e \in (\mathcal{A})_{\mathbf{j}}} |e|^k$$

$\{(\mathcal{A})_{\mathbf{j}}$ denotes a block indexed by multi-index $\mathbf{j}\}$

5: **end for**
6: **return** $\sqrt[k]{z}$

Let $(\mathcal{A})_{\mathbf{j}}$ be an off-diagonal block; hence $j_1 < j_2 < \ldots < j_d$. In order to compute the Froebenius norm, its elements must be counted $d!$ times in the sum since such a block appears—up to generalized transpositions—$d!$ times in the full super-symmetric tensor. Since the Froebenius norm is an element-wise function, the order of tensor elements is not important.

In the other two cases, partially diagonal or super-diagonal blocks have repeating indices—their multi-indices $\mathbf{j}$ are equal to

$$\begin{aligned}(j_1 < \ldots < \underbrace{j_{s_1} = \ldots = j_{s_{r_1}}}_{r_1} < \ldots < \\ < \underbrace{j_{s_2} = \ldots = j_{s_{r_2}}}_{r_2} < \ldots).\end{aligned} \tag{19}$$

Such blocks are repeated $d!/\prod_l r_l!$ times in the full tensor. Note that, if $j_1 < j_2 < \ldots < j_d$ then $\prod_l r_l! = 1$. In the super-diagonal case, i.e., $\mathbf{j} = (\underbrace{j_1 = \ldots = j_d}_{d})$, we have

$$\frac{d!}{\prod_l r_l!} = \frac{d!}{d!} = 1, \tag{20}$$

so the super-diagonal block is counted only once, as expected.

The advantage of Algorithm 5 is that it iterates over blocks in the block structure, which allows efficient

computation of internal sum elements. A naïve element-wise norm calculation approach would require n^d power operations. To compute $\|\mathcal{A}\|$ using Algorithm 5, we need b^d power operations for each block. Taking advantage of the block structure, the required number of multiplications can be approximated by $n^d/d!$. Finally, the computational complexity of Algorithm 5 is small compared with that of Algorithm 2; the complexity of the procedure of cumulant updates and computation of their norms can be approximated by Eqn. (15).

4.2. Data stream generation. In order to illustrate the functioning of the aforementioned algorithms, we use an artificially generated stream of data.

The initial data batch $\mathbf{X}^{(1)} \in \mathbb{R}^{t\times n}$ is sampled from a Gaussian multivariate distribution $\mathcal{N}(\mu, \Sigma)$, where ideally $\mu = \mathcal{C}_1(\mathbf{X}^{(1)})$, $\Sigma = \mathcal{C}_2(\mathbf{X}^{(1)})$. The subsequent data batches $\mathbf{X}^{(w)}_{(+)} \in \mathbb{R}^{t_{\text{up}}\times n}$, for $w \geq 1$ are sampled from a distribution F, which is discussed in what follows. Our goal is to determine if the distribution of the updated data $\mathbf{X}^{(w)} \in \mathbb{R}^{t\times n}$ did not change with growing w, and if it is still a multivariate Gaussian.

A naïve approach would be to compute multiple univariate statistics, such as, e.g., asymmetry κ_3 and kurtosis κ_4 for each of the marginal variables of the data stream in windows $\mathbf{X}^{(w)}$ (Gama, 2010). However, such an approach is oversimplified, since from the Sklar theorem (Sklar, 1959) one can deduce that it is always possible to construct a non-Gaussian multivariate distribution F that has all marginal distributions (F_i) being univariate Gaussian. Hence, despite $\forall_i\ \kappa_3(F_i) = 0, \kappa_4(F_i) = 0$, F is not a multivariate Gaussian.

To generate such data in practice, we can use a copula approach; see, *e.g.*, the work of Cherubini *et al.* (2004) for a definition and formal introduction of copulas. The probability distribution F is derived from the t-Student copula parametrized by Σ and ν as defined by Cherubini *et al.* (2004). In our case, we set the parameter to $\nu = 10$ degrees of freedom and the marginals equal to those of $\mathbf{X}^{(1)}$.

In order to visualize statistics of the generated data, we calculate the maxima over marginals of the absolute values of univariate asymmetries and kurtosises for $\mathbf{X}^{(w)}$. The results are presented in Fig. 5; as discussed before, neither univariate asymmetry nor kurtosis is significantly affected by the update.

4.3. Stream statistics analysis. In order to detect a change in the probability distribution, we calculate the following values of cumulant based measures in the function of w: $\|\mathcal{C}_2(\mathbf{X}^{(w)})\|$, $\nu_3(\mathbf{X}^{(w)})$ and $\nu_4(\mathbf{X}^{(w)})$, see Eqn. (18). The obtained results are gathered in Fig. 6. Analyzing Fig. 6(a), one can see that the norm of the covariance matrix is not significantly affected by

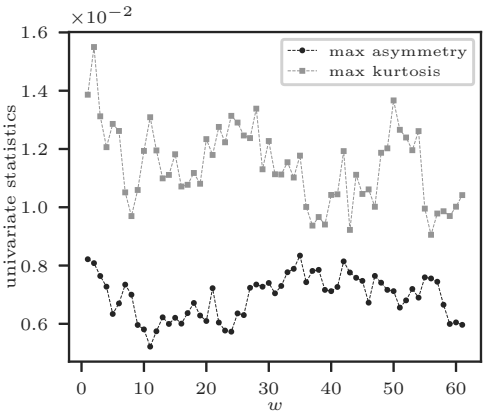


Fig. 5. Maximums of absolute values of univariate asymmetries and kurtosises for $\mathbf{X}^{(w)}$ with the number of marginal values $n = 60$, $t = 10^6$ data samples, $t_{\text{up}} = 2.5 \times 10^4$, $w_{\max} = 61$.

the updates, which is due to the particular choice of the t-Student copula parameters (Cherubini *et al.*, 2004) used to generate the updates.

Further, as presented in Fig. 6(b) $\nu_3(\mathbf{X}^{(w)})$ is also unaffected by updates, because the t-Student copula is symmetric (Cherubini *et al.*, 2004) in such a way that, given symmetric marginals, its high-order odd cumulants are zero. However, given the t-Student copula this is not the case for even cumulants, e.g., $\nu_4(\mathbf{X}^{(w)})$ is strongly affected by updates and in this case it can by used to distinguish between underlying distributions from which data are drawn. The values of $\nu_4(\mathbf{X}^{(w)})$ increase with an increasing window number w, up to $w = 41$, since for $w > 41$ there are no original data from the multivariate Gaussian distribution left in $\mathbf{X}^{(w)}$.

The normalization factor in the denominator of ν_4 assures that the function behaves similarly for different numbers of marginal variables n. This behavior depends on the particular choice of the t-Student copula used in updates; however, in general, the choice of the particular measure ν_d should depend on the expected statistical model of a data stream.

Let us discuss the approximation error of $\nu_4(\mathbf{X})$. We assume that the estimation error of the d-th cumulant elements comes mainly from the estimation error of the corresponding d-th moment element. In a super-diagonal case, we can refer directly to Appendix A in the work (Domino *et al.*, 2018b) and recall that the standard error of the estimation of the d-th univariate moment m_d is limited by $\sqrt{m_{2d}/t}$, in our case it is limited by $\sqrt{7!!/10^6} \approx 10^{-2}$. In a case of off-diagonal elements of $\mathcal{C}_4$, as mentioned in the aforementioned Appendix A, the estimation error is limited by the product of lower order moments which generally should by limited by m_{2d}, since the values of the moments grow rapidly with

d. Finally, while computing $||\mathcal{C}_4||$ (see (17)), we sum up the squares of its elements. Hence their individual errors should cancel out to some extent. However, the dependence between those elements is complex and a standard error calculus would be complicated. Hence we performed 100 numerical experiments, and computed ν_4 for $n = 100$ and $t = 10^6$ from generated data. We obtained the following results summarized by the triple of values: the 5th quantile, the median and the 95th quantile the of ν_4 values. At $w = 1$ (Gaussian multivariate distribution) we obtained $(0.004, 0.006, 0.011)$, while at $w \geq 41$ (t-Student copula with Gaussian marginals) we obtained $(0.199, 0.209, 0.220)$. In the second case, the error is higher since the t-Student copula introduces high order dependencies between data and elements of $\mathcal{C}_4$. Concluding, the estimation error is small in comparison with the values of ν_4.

4.4. Data frequency analysis. In order to estimate the maximal frequency of a data stream that can be analyzed on-line using Algorithm 1, we performed the following experiment using the same hardware as discussed in Section 3.2. We fixed the number of samples in a observation window t and varied the number of marginals n and the number of samples in a batch t_{up}. After Line 9 of Algorithm 1 had been executed, the values of $\|\mathcal{C}_1(\mathbf{X}^{(w)})\|$, $\|\mathcal{C}_2(\mathbf{X}^{(w)})\|$, $\nu_3(\mathbf{X}^{(w)})$ and $\nu_4(\mathbf{X}^{(w)})$ were calculated.

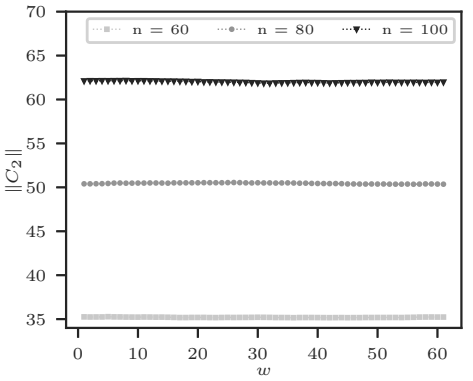
In Fig. 7 we present the maximal frequency of data analyzed on-line using the proposed scheme. In the presented example we computed and updated cumulants of orders $1, \ldots, 4$ and used $\|\mathcal{C}_1(\mathbf{X}^{(w)})\|$, $\|\mathcal{C}_2(\mathbf{X}^{(w)})\|$, $\nu_3(\mathbf{X}^{(w)})$ and $\nu_4(\mathbf{X}^{(w)})$ to extract statistical features.

Note that Algorithm 4 for updates of cumulants is independent of t_{up} and therefore has constant execution time. Hence one can increase the maximal data frequency at the expense of the sensitivity of the method by increasing t_{up}.

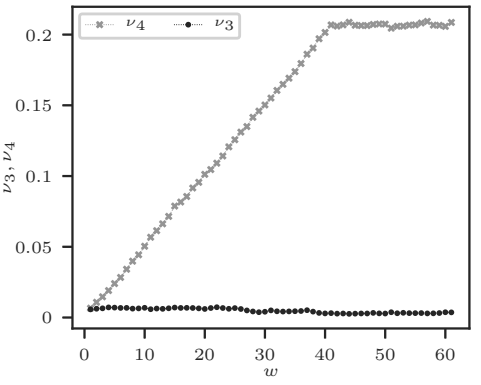
5. Conclusions

In this paper we have introduced a sliding window cumulant calculation algorithm for on-line processing high frequency multivariate data. For computer hardware described in Section 3, we have obtained a maximum data processing frequency of 150–2000 Hz depending on the number of marginal variables. We have presented an illustratory application of our algorithm by employing an example of Gaussian distributed data updated by data generated using the work on t-Student copula. We have shown that our algorithm can be used successfully to determine if on-line updates break the Gaussian distribution.

We believe that the presented algorithm can find many new applications, for example, in on-line signal



(a) $\|C_2\|$



(b) $\nu_3(\mathbf{X}^{(w)})$ $\nu_4(\mathbf{X}^{(w)})$ $n = 100$

Fig. 6. Cumulant based statistical measures for data $\mathbf{X}^{(w)} \in \mathbb{R}^{t \times n}$, with window width $t = 10^6$, update width $t_{\text{up}} = 2.5 \times 10^4$, and the number of windows $w_{\max} = 61$. The initial data are drawn from a Gaussian distribution; then the subsequent updates are drawn from a non-Gaussian one. We can observe that ν_4 is a good estimator of non-Gaussianity in contrast to function ν_3. The value of ν_4 increases with each window up to a saturation point at $w = 41$.

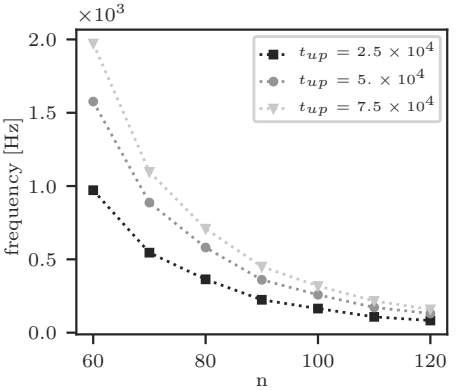


Fig. 7. Maximum frequency of on-line data analyzed using the cumulant update scheme: $d = 4, b = 4$; multiprocessing computation on 6 workers.

filtering or classification of data streams. The algorithm can be combined with many different methods of cumulant-based statistical features extractions, such as independent component analysis (ICA) (Blaschke and Wiskott, 2004; Virta *et al.*, 2015) or those based on tensor eigenvalues (Qi, 2005).

Acknowledgment

The research was partially financed by the National Science Centre in Poland (project no. 2014/15/B/ST6/05204). The authors would like to thank Przemysław Głomb for revising the manuscript and providing valuable comments.

References

Arismendi Zambrano, J. and Kimura, H. (2014). Monte Carlo approximate tensor moment simulations, *Numerical Linear Algebra with Applications*, DOI: 10.2139/ssrn.2491639.

Barndorff-Nielsen, O.E. and Cox, D.R. (1989). *Asymptotic Techniques for Use in Statistics*, Chapman & Hall, London/ New York, NY.

Becker, H., Albera, L., Comon, P., Haardt, M., Birot, G., Wendling, F., Gavaret, M., Bénar, C.-G. and Merlet, I. (2014). EEG extended source localization: Tensor-based vs. conventional methods, *NeuroImage* **96**: 143–157.

Bezanson, J., Chen, J., Karpinski, S., Shah, V. and Edelman, A. (2014). Array operators using multiple dispatch: A design methodology for array implementations in dynamic languages, *Proceedings of the ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming, Edinburgh, UK*, p. 56.

Bezanson, J., Edelman, A., Karpinski, S. and Shah, V.B. (2017). Julia: A fresh approach to numerical computing, *SIAM Review* **59**(1): 65–98.

Bezanson, J., Karpinski, S., Shah, V.B. and Edelman, A. (2012). Julia: A fast dynamic language for technical computing, *arXiv:*1209.5145.

Birot, G., Albera, L., Wendling, F. and Merlet, I. (2011). Localization of extended brain sources from EEG/MEG: The ExSo-MUSIC approach, *NeuroImage* **56**(1): 102–113.

Blaschke, T. and Wiskott, L. (2004). Cubica: Independent component analysis by simultaneous third-and fourth-order cumulant diagonalization, *IEEE Transactions on Signal Processing* **52**(5): 1250–1256.

Cherubini, U., Luciano, E. and Vecchiato, W. (2004). *Copula Methods in Finance*, John Wiley & Sons, Chichester.

Chevalier, P., Ferréol, A. and Albera, L. (2006). High-resolution direction finding from higher order statistics: The $2rmq$-MUSIC algorithm, *IEEE Transactions on Signal Processing* **54**(8): 2986–2997.

Comtet, L. (1974). *Advanced Combinatorics,* Reidel Pub., Boston, MA.

Domino, K. (2017). The use of the multi-cumulant tensor analysis for the algorithmic optimisation of investment portfolios, *Physica A: Statistical Mechanics and Its Applications* **467**: 267–276.

Domino, K. and Gawron, P. (2018). CumulantsUpdates.jl, *Zenodo*, DOI:10.5281/zenodo.1213205.

Domino, K., Gawron, P. and Pawela, Ł. (2018a). Cummulants.jl, *Zenodo*, DOI:10.5281/zenodo.1185137.

Domino, K., Pawela, Ł. and Gawron, P. (2017). SymmetricTensors.jl, *Zenodo*, DOI: 10.5281/zenodo.996222.

Domino, K., Pawela, Ł. and Gawron, P. (2018b). Efficient computation of higer-order cumulant tensors, *SIAM Journal on Scientific Computing* **40**(3): A1590–A1610.

Gama, J. (2010). *Knowledge Discovery from Data Streams*, Chapman & Hall/CRC Data Mining and Knowledge Discovery Series, Vol. 20103856, Chapman and Hall/CRC, Boca Raton, FL.

Geng, M., Liang, H. and Wang, J. (2011). Research on methods of higher-order statistics for phase difference detection and frequency estimation, *4th International Congress on Image and Signal Processing (CISP), Shanghai, China*, Vol. 4, pp. 2189–2193.

Graham, R.L., Knuth, D.E. and Patashnik, O. (1989). *Concrete Mathematics: A Foundation for Computer Science*, Addison & Wesley, Reading, MA.

Hyvärinen, A. (2014). Independent component analysis of images, *in* D. Jaeger and R. Jung (Eds.), *Encyclopedia of Computational Neuroscience*, Springer, New York, NY, pp. 1–5.

Jondeau, E., Jurczenko, E. and Rockinger, M. (2018). Moment component analysis: An illustration with international stock markets, *Journal of Business & Economic Statistics* **36**(4): 576–598, DOI: 10.1080/07350015.2016.1216851.

Kendall, M.G. (1946). *The Advanced Theory of Statistics*, 2nd Edn., Charles Griffin and Co., London.

Knuth, D.E. (2011). *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*, Addison-Wesley Professional, Boston, MA.

Latimer, J.R. and Namazi, N. (2003). Cumulant filters—a recursive estimation method for systems with non-Gaussian process and measurement noise, *Proceedings of the 35th Southeastern Symposium on System Theory, Morgantown, WV, USA*, pp. 445–449.

Lukacs, E. (1970). *Characteristics Functions*, Griffin, London.

Martin, I.W. (2013). Consumption-based asset pricing with higher cumulants, *The Review of Economic Studies* **80**(2): 745–773.

Qi, L. (2005). Eigenvalues of a real supersymmetric tensor, *Journal of Symbolic Computation* **40**(6): 1302–1324.

Rubinstein, M., Jurczenko, E. and Maillet, B. (2006). *Multi-Moment Asset Allocation and Pricing Models*, Vol. 399, John Wiley & Sons, Chichester.

Schatz, M.D., Low, T.M., van de Geijn, R.A. and Kolda, T.G. (2014). Exploiting symmetry in tensors for high performance: Multiplication with symmetric tensors, *SIAM Journal on Scientific Computing* **36**(5): C453–C479.

Sklar, A. (1959). *Fonctions de répartition à n dimensions et leurs marges*, Publications de l'Institut de Statistique de l'Université de Paris, Paris.

Stefanowski, J., Krawiec, K. and Wrembel, R. (2017). Exploring complex and big data, *International Journal of Applied Mathematics and Computer Science* **27**(4): 669–679, DOI: 10.1515/amcs-2017-0046.

Virta, J., Nordhausen, K. and Oja, H. (2015). Joint use of third and fourth cumulants in independent component analysis, *arXiv:* 1505.02613.

Krzysztof Domino was born in 1982. He received his MSc and PhD degrees, both in physics, respectively in 2006 from Jagiellonian University in Kraków and in 2015 from Silesian University in Katowice. In the meantime (2006–2014) he was working in industry. Since 2015 he has been with the Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, currently as an assistant professor. He is a member of the Quantum Systems of Informatics Group therein. His research interests include multivariate non-Gaussian statistics, high order statistics, auto-correlation analysis and quantum computation.

Piotr Gawron works in the Quantum Systems of Informatics Group of the Institute of Theoretical and Applied Informatics, Polish Academy of Sciences. He holds an MSc in informatics (2003, Silesian University of Technology). He received his PhD in informatics (2008) at the Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, and his DSc degree in technical sciences in the field of informatics (2014) at Faculty of Automatic Control, Electronics and Computer Science, Silesian University of Technology.

Received: 5 April 2018
Revised: 2 July 2018
Accepted: 3 October 2018