Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

A fast algorithm for computing a matrix transform used to detect trends in noisy data[☆,☆☆]

Dan Kestner[a,*], Glenn Ierley[b], Alex Kostinski[a]

[a] Physics Department, Michigan Technological University, 1400 Townsend Dr, Houghton, MI 49931, United States of America
[b] Mathematics Department, Michigan Technological University, 1400 Townsend Dr, Houghton, MI 49931, United States of America

ARTICLE INFO

Article history:
Received 31 January 2020
Received in revised form 25 April 2020
Accepted 4 May 2020
Available online 19 May 2020

Keywords:
Computational statistical methods
Numerical linear algebra
Numerical optimization
Noise
Computer data analysis and implementation

ABSTRACT

A recently discovered universal rank-based matrix method to extract trends from noisy time series is described in Ierley and Kostinski (2019) but the formula for the output matrix elements, implemented there as an open-access supplement MATLAB computer code, is $\mathcal{O}(N^4)$, with N the matrix dimension. This can become prohibitively large for time series with hundreds of sample points or more. Based on recurrence relations, here we derive a much faster $\mathcal{O}(N^2)$ algorithm and provide code implementations in MATLAB and in open-source JULIA. In some cases one has the output matrix and needs to solve an inverse problem to obtain the input matrix. A fast algorithm and code for this companion problem, also based on the recurrence relations, are given. Finally, in the narrower, but common, domains of (i) trend detection and (ii) parameter estimation of a linear trend, users require, not the individual matrix elements, but simply their accumulated mean value. For this latter case we provide a yet faster $\mathcal{O}(N)$ heuristic approximation that relies on a series of rank one matrices. These algorithms are illustrated on a time series of high energy cosmic rays with $N > 4 \times 10^4$.

Program summary

Program Title: Pfromdata, QofP, mbasisandcoeffs, nonzerop, Qavgapprox, PofQ, mexact, CodeTesting
CPC Library link to program files: http://dx.doi.org/10.17632/mkcxrky9jc.1
Licensing provisions: MIT
Programming language: MATLAB and Julia
Nature of problem: An order-rank data matrix and its transform to a stable form are used repeatedly to detect and/or extract trends from noisy data. An efficient yet accurate calculation of the matrix transform is therefore required.
Solution method: We introduce and apply an analytic recursion relation, which speeds up the execution of the matrix transform from $\mathcal{O}(N^4)$ arithmetic operations to $\mathcal{O}(N^2)$. Since this matrix transform is called often during optimization, our improvement allows for far shorter optimization times, for a given sample size. For example, a transform whose time is extrapolated to an unrealistic 75 days on a Dell personal laptop computer with a 2.2 GHz quad-core AMD processor running 32 bit MATLAB version R2015b on 64 bit Windows 10 ($N = 5000$), now takes a fraction of a second.

References

[1] Universal Rank-Order Transform to Extract Signals from Noisy Data, Glenn Ierley and Alex Kostinski, Phys. Rev. X 9 031039 (2019).

1. Introduction

A broadly-applicable rank-based approach for detection and extraction of generally non-linear trends in noisy time series has recently been introduced [4] and we shall now briefly review the mathematical essentials. The input time series is segmented into n_t samples, with each sample having n_T data points. A square $n_T \times n_T$ population matrix P is then calculated such that $P_{j,k}$ is the population (number) of data points with order j (position in the sample), and rank k (position in the sample after an ascending sort) [4]. Alternatively, $P_{j,k}$ can also be viewed as a 2D probability density function (pdf) or a histogram over the plane defined by rank and times axes. The matrix P is illustrated below in (1).

[☆] The review of this paper was arranged by Prof. Hazel Andrew.
[☆☆] This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).
[*] Corresponding author.
E-mail address: djkestne@mtu.edu (D. Kestner).

https://doi.org/10.1016/j.cpc.2020.107382
0010-4655/© 2020 Elsevier B.V. All rights reserved.

$$P = \left(\begin{array}{ccccc|ccc}
P_{1,1} & P_{1,2} & \cdots & P_{1,k-1} & P_{1,k} & P_{1,k+1} & \cdots & P_{1,n_T} \\
P_{2,1} & P_{2,2} & \cdots & P_{2,k-1} & P_{2,k} & P_{2,k+1} & \cdots & P_{2,n_T} \\
\vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\
P_{j-1,1} & P_{j-1,2} & \cdots & P_{j-1,k-1} & P_{j-1,k} & P_{j-1,k+1} & \cdots & P_{j,n_T} \\
P_{j,1} & P_{j,2} & \cdots & P_{j,k-1} & P_{j,k} & P_{j,k+1} & \cdots & P_{j,n_T} \\
\hline
P_{j+1,1} & P_{j+1,2} & \cdots & P_{j+1,k-1} & P_{j+1,k} & P_{j+1,k+1} & \cdots & P_{k+1,n_T} \\
\vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\
P_{n_T,1} & P_{n_T,2} & \cdots & P_{n_T,k-1} & P_{n_T,k} & P_{n_T,k+1} & \cdots & P_{n_T,n_T}
\end{array}\right) \tag{1}$$

To “zoom in” on the trends hidden in P, the Q-transform was introduced [4] as follows

$$Q_{j,k} = \left(\frac{\sum_{m=1}^{j}\sum_{n=1}^{k} P_{m,n} + \sum_{m=j+1}^{n_T}\sum_{n=k+1}^{n_T} P_{m,n}}{jk + (n_T - j)(n_T - k)} - \frac{\sum_{m=1}^{j}\sum_{n=k+1}^{n_T} P_{m,n} + \sum_{m=j+1}^{n_T}\sum_{n=1}^{k} P_{m,n}}{(n_T - j)k + j(n_T - k)} \right) \frac{n_T}{n_t} \tag{2}$$

To understand the construction, consider the division of P into quadrants for calculation of $Q_{j,k}$ as shown on the RHS of (1). Each element of Q is the difference between the average matrix element of the combined upper left and lower right quadrants of P, and the average matrix element of the combined upper right and lower left quadrants, normalized by the overall average matrix element of P, $\langle P \rangle \equiv \sum_{m=1}^{n_T}\sum_{n=1}^{n_T} P_{m,n}/n_T^2 = n_t/n_T$.

The number of operations $(+, -, \times, \div)$ required to compute Q using Eq. (2) is $\mathcal{O}(n_T^4)$. For large n_T and repeated calls, as will often be needed in applications, the computation time can become prohibitively long. In addition, setting $\langle Q \rangle = 0$ where angular brackets denote average over matrix elements, functions as a trend detector when the functional form of the trend is not available and we shall illustrate it on the time series of cosmic rays in 5. To that end, our purpose in this paper is four-fold:

(i) present a $\mathcal{O}(n_T^2)$ algorithm for computing the Q-transform and its MATLAB implementation;

(ii) supply an open source (Julia) implementation;

(iii) present an efficient $\mathcal{O}(n_T)$ calculation of $\langle Q \rangle$, where $\langle Q \rangle$ is the average matrix element of Q. The departure of this (scalar) quantity from zero is used to detect presence of trend [4].

(iv) provide an illustrative example from a long cosmic ray time series;

To provide a point of reference for (i) and (iii), we compare to 2D convolution, which is numerically comparable to the Q-transform. We find that: (1) our $\mathcal{O}(n^2)$ scaling for (i) matches 2D convolution with a small (3 × 3, 4 × 4, etc.) mask (2) our $\mathcal{O}(n)$ scaling for (iii) matches the scaling of an approximate 2D convolution, which similarly to (iii) uses a low rank approximation [3].

2. Derivation of the algorithm

To begin, note that Eq. (2) can be simplified by making use of constraints on P that each row and column sum to n_t. Thus, the sums of elements in the four quadrants of P, entering the numerator of (2) are not independent. Numbering the quadrants as 1–4 beginning from the upper right, moving counterclockwise, and calling the sums of elements in each quadrant i as $\Sigma_{j,k}^{(i)}$, we have

$$\begin{aligned}
\Sigma_{j,k}^{(1)} &= \sum_{m=1}^{j}\sum_{n=k+1}^{n_T} P_{m,n} \\
\Sigma_{j,k}^{(2)} &= \sum_{m=1}^{j}\sum_{n=1}^{k} P_{m,n} \\
\Sigma_{j,k}^{(3)} &= \sum_{m=j+1}^{n_T}\sum_{n=1}^{k} P_{m,n} \\
\Sigma_{j,k}^{(4)} &= \sum_{m=j+1}^{N_T}\sum_{n=k+1}^{n_T} P_{m,n}
\end{aligned} \tag{3}$$

Their dependence on each other are expressed as follows:

$$\begin{aligned}
\Sigma_{j,k}^{(1)} + \Sigma_{j,k}^{(2)} &= jn_t \\
\Sigma_{j,k}^{(2)} + \Sigma_{j,k}^{(3)} &= kn_t \\
\Sigma_{j,k}^{(3)} + \Sigma_{j,k}^{(4)} &= (n_T - j)n_t \\
\Sigma_{j,k}^{(4)} + \Sigma_{j,k}^{(1)} &= (n_T - k)n_t
\end{aligned} \tag{4}$$

This system of four equations in four unknowns ($\Sigma_{j,k}^{(i)}$, $i = 1-4$) is under-determined and when recast as a 4 × 4 matrix equation, has a matrix rank of three. Thus, only one of the four $\Sigma_{j,k}^{(i)}$ is independent and we picked $\Sigma_{j,k}^{(2)}$ for that purpose.

$$\begin{aligned}
\Sigma_{j,k}^{(1)} &= jn_t - \Sigma_{j,k}^{(2)} \\
\Sigma_{j,k}^{(2)} &= \Sigma_{j,k}^{(2)} \\
\Sigma_{j,k}^{(3)} &= kn_t - \Sigma_{j,k}^{(2)} \\
\Sigma_{j,k}^{(4)} &= (n_T - j - k)n_t + \Sigma_{j,k}^{(2)}
\end{aligned} \tag{5}$$

This can be substituted back into Eq. (2),

$$Q_{j,k} = \left(\frac{(n_T - j - k)n_t + 2\Sigma_{j,k}^{(2)}}{jk + (n_T - j)(n_T - k)} - \frac{(j + k)n_t - 2\Sigma_{j,k}^{(2)}}{(n_T - j)k + j(n_T - k)} \right) \frac{n_T}{n_t} \tag{6}$$

Define D as a $(n_T - 1) \times (n_T - 1)$ matrix, whose elements are the product of the two denominators in Eq. (2):

$$D_{j,k} = (jk + (n_T - j)(n_T - k))(j(n_T - k) + (n_T - j)k) \tag{7}$$

The Q matrix can be expressed compactly in terms of $\Sigma_{j,k}^{(2)}$ and $D_{j,k}$.

$$Q_{j,k}D_{j,k} = \frac{2n_T^3}{n_t}\left(\Sigma_{j,k}^{(2)} - \frac{n_t}{n_T}jk\right) \tag{8}$$

The motivation for this is that the second quadrant sum $\Sigma_{j,k}^{(2)}$ satisfies a recurrence relation.

$$\Sigma_{j,k}^{(2)} = \Sigma_{j-1,k}^{(2)} + \Sigma_{j,k-1}^{(2)} - \Sigma_{j-1,k-1}^{(2)} + P_{j,k} \tag{9}$$

Taken together with Eq. (8), this yields a recurrence relation for Q.

$$Q_{j,k} = \frac{1}{D_{j,k}}\left(D_{j,k-1}Q_{j,k-1} + D_{j-1,k}Q_{j-1,k} - D_{j-1,k-1}Q_{j-1,k-1} + \frac{2n_T^3}{n_t}(P_{j,k} - \frac{n_t}{n_T}) \right) \tag{10}$$

The algorithm used to calculate Q via (10) is described in Algorithm 1 and its MATLAB and Julia implementations accompany this manuscript. The full Q matrix is calculable in $\mathcal{O}(n_T^2)$

operations, as is seen by observing that each element of Q can be calculated in $\mathcal{O}(1)$ from a small number of neighboring Q elements and some constants, and that the total number of elements in Q is $(n_T-1)^2$.

Algorithm 1: An $\mathcal{O}(n_T^2)$ implementation of the Q-transform, using recurrence in Eq. (10). The (1,1) element is found first. Rows and columns are found by moving rightwards or downwards from diagonal elements. All elements are found from neighboring elements to the left, above-left, and above.

```
Data: $n_T \times n_T$ population matrix $P$, satisfying row and column
    sum constraints
Result: $(n_T-1)\times(n_T-1)$ matrix $Q$
for $i = 1$ to $n_T - 1$ do
    $Q_{i,i} \leftarrow P_{i,i}, D_{i,i}, Q_{i-1,i}D_{i-1,i}, Q_{i,i-1}D_{i,i-1}, Q_{i-1,i-1}D_{i-1,i-1}$;
    for $m = i+1$ to $n_T - 1$ do
        $Q_{i,m} \leftarrow$
        $P_{i,m}, D_{i,m}, Q_{i-1,m}D_{i-1,m}, Q_{i,m-1}D_{i,m-1}, Q_{i-1,m-1}D_{i-1,m-1}$;
        $Q_{m,i} \leftarrow$
        $P_{m,i}, D_{m,i}, Q_{m-1,i}D_{m-1,i}, Q_{m,i-1}D_{m,i-1}, Q_{m-1,i-1}D_{m-1,i-1}$;
    end
end
```

3. Analytical results

One key result of this paper is Eq. (10), just derived. This permits an $\mathcal{O}(n_T^2)$ method for calculating Q that is much faster than the $\mathcal{O}(n_T^4)$ brute force evaluation of Eq. (2), especially for large n_T. Another essential result is the transformation for P, given Q. This was obtained by rearranging Eq. (10) as follows.

$$P_{j,k} = \Big(D_{j,k}Q_{j,k} - D_{j,k-1}Q_{j,k-1} - D_{j-1,k}Q_{j-1,k} + D_{j-1,k-1}Q_{j-1,k-1}\Big)\frac{n_t}{2n_T^3} + \frac{n_t}{n_T} \tag{11}$$

Not only does this transformation turn out to be stably computable but also efficiently so. In fact, it can be accomplished also in $O(n_T^2)$ operations and is implemented in MATLAB and Julia programs in the accompanying files. These results allow analyses of previously inaccessible data because of the prohibitively long computation times. The confirmation of the speed up of Eq. (10) over Eq. (2) directly in terms of CPU time is given in Fig. 1.

As a sample application, possible because of the computational improvement provided by calculating the Q-transform recursively rather than by the direct evaluation of the double sums in Eq. (2), we choose $n_T = 5000 \approx 2^{12.3}$, which lies just outside of the axis range shown in Fig. 1. A calculation using the recursive result in Eq. (10) takes about 0.7 s [1]. In comparison, using Fig. 1 to extrapolate the $O(n_T^4)$ curve out to $\log_2(n_T) = 12.3$, a brute force calculation would take approximately 80 days and hence is not shown in the figure.

4. Fast algorithm for calculating $\langle Q \rangle$

We now turn to efficient calculation of $\langle Q\rangle$, the mean matrix element of Q in the special case of large n_T and small n_t. For example, the single sample (one time series) cases, the matrix P is sparse, consisting of ${N_T}^2 - N_T$ zeros and need not be stored in memory in its entirety. Rather, only indices of the non-zero matrix elements, found by independently sorting each of the repeated n_t trials, may suffice to calculate Q. Also, it was shown in [4] that the entire Q information is not required when one is concerned merely with trend detection or parameter estimation of a linear trend. One such application (time series of cosmic ray arrivals) is discussed in the next section. In such cases, it suffices to calculate only element-averaged metric

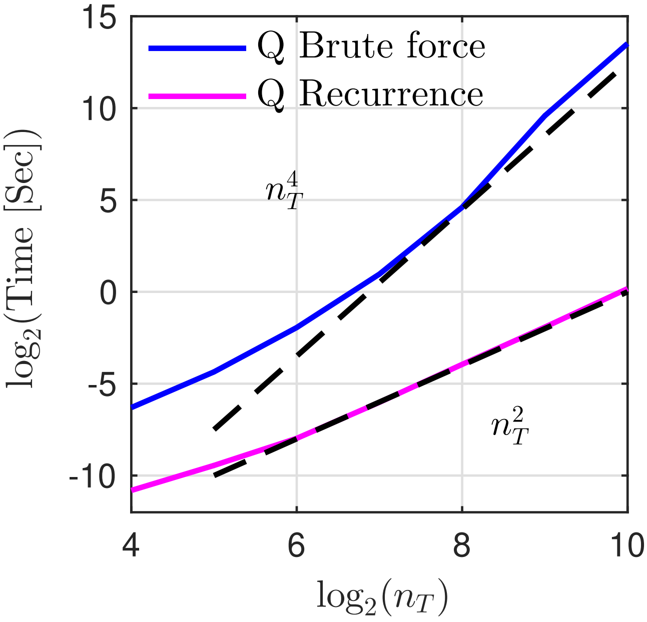

Fig. 1. A comparison of calculation times for the Q-transform. A recursive approach reduces the CPU time from $\mathcal{O}(n_T^4)$ to $\mathcal{O}(n_T^2)$. The blue curve shows results from an implementation of the brute force approach of Eq. (2), while the red curve shows timing data from the optimized calculation of Eq. (10). The contrast in complexity order becomes apparent for larger size matrices, $\log_2(n_T) \gtrapprox 4$. Note that the calculation time becomes intractable for $n_T \gtrapprox 2^{10} \approx 1024$ for the brute force calculation [1]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$$\langle Q\rangle \equiv \frac{1}{(n_T-1)^2}\sum_{j,k=1}^{n_T-1} Q_{j,k} \tag{12}$$

A sufficiently large value of $\langle Q\rangle$ implies the presence of a trend. Here, "sufficiently large" is with reference to a fiducial value expected for pure noise, a formula for which in terms of n_T and n_t is given in [4]. Using the results of Section 3, Q, and thus $\langle Q\rangle$ (which has $(n_T-1)^2$ terms), can be computed from P in $\mathcal{O}(n_T^2)$ operations. In order to calculate $\langle Q\rangle$, only accumulated mean value is needed and the rest of the matrix elements need not be stored, thereby reducing complexity of the calculation. To that end, as is shown in Appendix B.3 of [4], since Eq. (2) is a linear map between elements of P and Q, it may be expressed as a matrix equation,

$$\mathbf{q} = M\mathbf{p} \tag{13}$$

where $\mathbf{q}$ and $\mathbf{p}$ are $(n_T-1)^2 \times 1$ and $n_T^2 \times 1$ column vector versions of P and Q, and M is $(n_T-1)^2 \times n_T^2$. Thus, $\langle Q\rangle$ can be written as,

$$\begin{aligned}\langle Q\rangle &= \frac{1}{(n_T-1)^2}\underbrace{[111...]}_{(n_T-1)^2}\mathbf{q} \\ &= \frac{1}{(n_T-1)^2}[111...]M\mathbf{p} \\ &= \mathbf{m}^T\mathbf{p}\end{aligned} \tag{14}$$

where,

$$\mathbf{m} = \frac{1}{(n_T-1)^2}M^T[111...]^T \tag{15}$$

Writing the $n_T^2 \times 1$ vector $\mathbf{m}$ as an $n_T \times n_T$ matrix m, we can express $\langle Q \rangle$ as the sum of elements of the Hadamard (element-wise) product of m and P itself.

$$\langle Q \rangle = \sum_{j,k} m_{j,k} P_{j,k} \tag{16}$$

The equation for m can now be expressed. Comparing Eqs. (15) and (13) with Eq. (2) and carefully converting between matrix indices and linear vector indices, we find an expression for m,

$$m_{j,k} = \frac{n_T}{n_t(n_T-1)^2}\left(\sum_{m=1}^{j-1}\sum_{n=1}^{k-1} d^{(1)}_{m,n} + \sum_{m=j}^{n_T-1}\sum_{n=k}^{n_T-1} d^{(1)}_{m,n} - \sum_{m=1}^{j-1}\sum_{n=k}^{n_T-1} d^{(2)}_{m,n} - \sum_{m=j}^{n_T-1}\sum_{n=1}^{k-1} d^{(2)}_{m,n} \right) \tag{17}$$

where

$$d^{(1)}_{m,n} = \frac{1}{mn + (n_T - m)(n_T - n)} \qquad d^{(2)}_{m,n} = \frac{1}{m(n_T - n) + (n_T - m)n} \tag{18}$$

The matrix m is not determined by the data, and depends solely on n_t and n_T, the former being only an inverse multiplicative constant. Once m is constructed, the mean element of Q is easily accessed from Eq. (16). For sparse P, $n_t \ll n_T$, this means a calculation of order n_T, much faster than the $\mathcal{O}(n_T^2)$ needed to calculate the matrix Q explicitly prior to averaging. As it stands, m takes $\mathcal{O}(n_T^4)$ to construct, which can become prohibitive for large n_T. Not only this, but m is memory limited to about $n_T = \sqrt{10} \times 10^4$ for an 8 GB RAM, being of type double (8 bytes per element). Since for $n_t \ll n_T$ the matrix P is sparse, in this case we only need calculate the elements of m corresponding to nonzeros in P. This greatly reduces demands on memory, allowing n_T, the number of data points per trial, to be as large as about 10^9 for $n_t = 1$ and a 8 GB RAM. Also, there is an $\mathcal{O}(n_T)$ way to approximate any element of m in $\mathcal{O}(1)$, operations, reducing the calculation of m for sparse P to $\mathcal{O}(n_T)$. For nonsparse P, when the full matrix m is needed, the approximation scheme gives m in $\mathcal{O}(n_T^2)$, due the number of elements needed. We also find that m may be calculated exactly, up to accumulated rounding errors due to finite machine precision, by an $\mathcal{O}(n_T^2)$ recursive algorithm, much like for Q in Section 2. The advantage in this case is that m need only be calculated once, for each n_T, and then it applies to any dataset of the same size parameter n_T, modulo a rescaling due to varying n_t. This saves the time needed for calculating Q itself each time. The disadvantage of using recursion to find m as compared with using the approximate approach is that recursion can only create contiguous rectangular blocks of m. In contrast, the approximation for m allows only the needed elements to be calculated, regardless of how they are spatially related in the matrix. To this we now turn.

To describe how the matrix m is approximated, which is the most efficient way to calculate $\langle Q \rangle$ for large n_T and $n_t \ll n_T$, we first note the following: the rank one matrix that is an outer product of the first column of m with itself, normalized by $1/m(1,1)$, provides a fair approximation of the entire matrix m. Seeing that this approximation is rank one suggests the approximation can be improved by adding another rank one term. This turns out to be so. One simply takes a linear combination of the first two columns of m, and since the first row and column of m are already exact, chooses this combination such that a new vector is obtained with vanishing first element. The outer product of this vector with itself is zero along the first row and column, and thus does not alter the previous exact rank one approximation there. If one then adds this new rank one matrix to the old, while choosing a scalar coefficient to match any of the elements in the original second column of m, one obtains a rank two approximation of m that is exact in the first *two* rows and columns. This holds true for any symmetric matrix, as a little algebra can show. Moreover, this extends easily: a third "basis vector" may be obtained with vanishing first and second elements by judiciously mixing the first three exact columns of m. Again adding the outer product of this new vector with itself to the rank 2 approximation, with an appropriate constant chosen to match an arbitrary element of the exact third column of m, a rank 3 approximation is obtained that is exact in the first three rows and columns. And so on. By generalization, beginning with r columns of m yields a rank r approximation, exact in the first r rows and columns of m. Since m always has the same form when regarded as a two dimensional function, the effect of increasing the number of rows/columns n_T is to bring nearby columns closer numerically. Therefore, practically, difficulties arise with the above approximation scheme due to nearby columns of m becoming linearly dependent for large n_T, but these can be circumvented by avoiding successive columns, but picking columns increasingly separated with n_T, so as to roughly maintain proportionate horizontal locations in the matrix m. It also proves advantageous to mix the columns such that the zeros in the new column vectors are also spaced out proportionately within the matrix and not merely adjacent and at the beginning. Letting $\mathbf{x_1}, \ldots \mathbf{x_r}$ be the selected exact columns of m, and $\mathbf{v_1}, \ldots \mathbf{v_r}$ the new vectors,

$$\mathbf{v_i} = (\mathbf{x_1}|\mathbf{x_2}\ldots|\mathbf{x_r}) \begin{pmatrix} 1 \\ a_i(1) \\ \vdots \\ a_i(r-1) \end{pmatrix} \tag{19}$$

where the coefficients $a_i(1), \ldots, a_i(r-1)$ are given by the requirement,

$$(\mathbf{v_i}(1), \mathbf{v_i}(1+s), \ldots, \mathbf{v_i}(1+(i-2)s))^T = (0, 0, \ldots, 0)^T \tag{20}$$

Heuristically, the optimal case is when the zeros in the new column vectors have the same spacing as the columns. This improves the conditioning of a certain matrix that must be inverted in this process. Also, we find that for uniform column spacing, there is an optimal rank approximation of $r = 6$. In this case, the optimal column/row zero spacings s are given empirically by $s = [0.92143543 + 0.02465247 \times n_T]$. Taking non-uniformly spaced columns of the matrix m yields generally much better results, as found for example by using MATLAB's Genetic Algorithm to find the optimal columns of m for a rank $r = 6$ expansion with row zero spacings matching the column spacings. We also choose the coefficients of the rank 1 matrices in order to optimize the approximation of m along its diagonal, via least squares (MATLAB's backslash operator).

Mathematically, the above can be summarized by saying that we approximate m with a handful (r, the rank of the approximation) of its columns,

$$m \approx m_1 + m_2 + m_3 + \cdots + m_r \tag{21}$$

The rank 1 matrices m_i are outer products of column vectors with themselves,

$$m_i = \mathbf{v_i}\mathbf{v_i^T} \tag{22}$$

The vectors $\mathbf{v_i}, i = 1, \ldots, r$ are linear combinations of r different columns of the exact m, calculated from Eq. (17), using the speedups of Eqs. (A.1) and (A.2). This is thus an approximation of the full column space of m. The generation of the approximate m is shown in Algorithm 2

Algorithm 2: Algorithm for approximation of m. A subset of columns of m are calculated exactly, then linearly combined to form a new set of basis vectors. Basis vectors form rank 1 matrices, with coefficients that minimize error along the full diagonal of m.

```
input : $n_T$, the size of (square) matrix $m$
output: Low rank approximation to $m$
approximation rank $r \leftarrow \min(6, \lfloor \frac{n_T}{2} \rfloor)$;
exact column spacing $s \leftarrow round(0.92143543 + 0.02465247 n_T)$;
$\mathbf{x_1}, \dots \mathbf{x_r} \leftarrow$ sample exact columns of $m$, columns 1, 1+s,...1+(r-1)s;
$\mathbf{v_1} \leftarrow \mathbf{x_1}$;
for i=2 to r do
    $\mathbf{v_i} \leftarrow$ linearly combine $\mathbf{x_1}, \dots \mathbf{x_i}$ to make elements 1, 1+s,...,1+(i-2)s be zero
end
calculate diagonal of $m$ using equations Eq. (A.1) and Eq. (A.3);
$c_1, \dots, c_r \leftarrow \min[(diag(c_1 \mathbf{v_1}\mathbf{v_1}^T + c_2 \mathbf{v_2}\mathbf{v_2}^T + \dots + c_r \mathbf{v_r}\mathbf{v_r}^T) - diag(m))^2]$;
return $\mathbf{v_1}, \dots \mathbf{v_r}, c_1, \dots, c_r$;
```

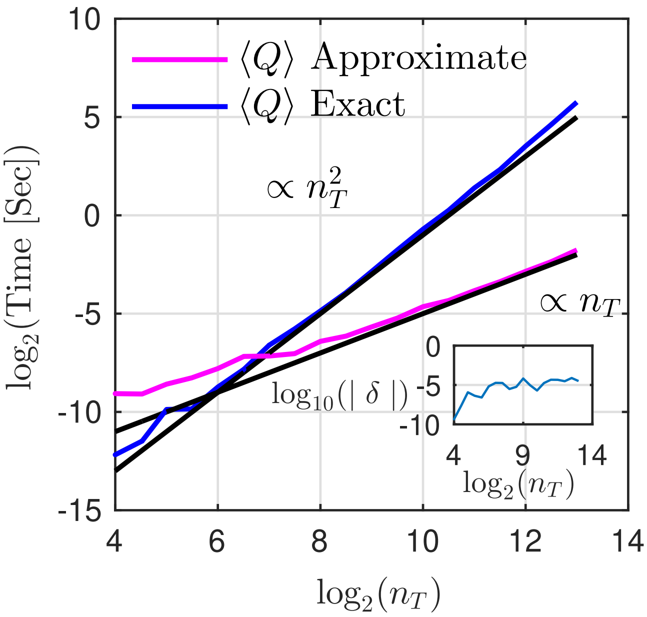

Fig. 2. $\mathcal{O}(n_T)$ vs. $\mathcal{O}(n_T^2)$ calculations of $\langle Q \rangle \equiv$ average over all elements of Q, in the limit $n_t \ll n_T$. The relative difference $\delta \equiv (\langle Q \rangle_{approx.} - \langle Q \rangle_{exact})/\langle Q \rangle_{exact}$ is shown in the inset. The linear time includes the time to "precompute" the needed subset of elements of the matrix m (see the text) and take the Hadamard product with P. The quadratic time includes the time to calculate, store, and average the full Q matrix.

We note that this approximation is essentially a low rank matrix approximation that uses low rank matrix completion, topics that both arise in data science [5].

With this approximation of m, we now show that the cost of computing m is reduced from $\mathcal{O}(n_T^2)$ to $\mathcal{O}(n_T)$ for the overhead, and $\mathcal{O}(1)$ operations for each element after that. We also show that the memory overhead is also $\mathcal{O}(n_T)$, plus the cost of each element computed after that (between $\mathcal{O}(n_T)$ and $\mathcal{O}(n_T^2)$).

Since only the non-zero entries of P contribute to the element-wise product with m, and P can have as few as $\mathcal{O}(n_T)$ non-zero entries (when $n_t = 1$), $\langle Q \rangle$ can be computed in as low as $\mathcal{O}(n_T)$ operations, after overhead that is also $\mathcal{O}(n_T)$, leaving a grand total of $\mathcal{O}(n_T)$ operations. This is in contrast to the $\mathcal{O}(n_T^2)$ operations it would take to compute Q using the recursive algorithm of Section 2, and then sum and average the elements of Q. This difference is illustrated in Fig. 2.

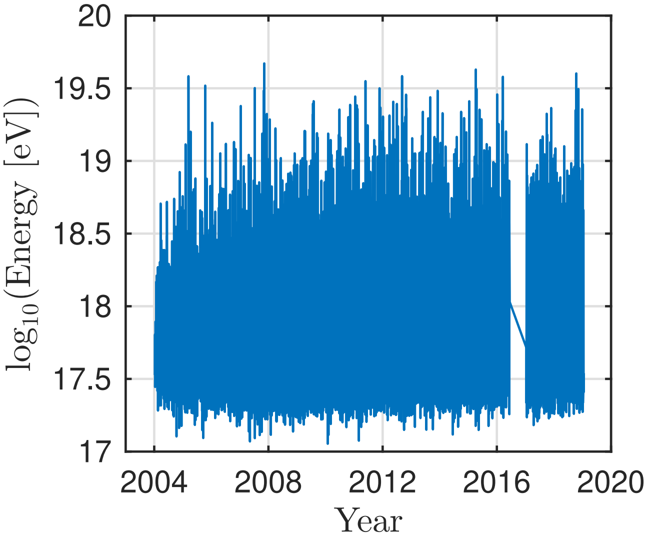

Fig. 3. Time series of energies of high energy cosmic rays. There are a total of 49223 events between 2004 and 2019. Energy scale is 10^{18}eV = 1EeV. The data is uncorrelated (white), and has a non Gaussian distribution.
Source: Data is taken from the Pierre Auger Observatory Public Event Explorer [2], and represents 1% of all data taken by the observatory.

5. Illustration on time series of cosmic rays

To illustrate the importance of the numerical acceleration for trend detection as just described in the previous section, we pick an example from cosmic ray physics. The data consists of 49,223 events (only 1% of the total data is available to general public), in a form of a time series of arrivals with various energies (see Fig. 3 from data in [2]).

Energy-resolved flux (spectrum) plays the central role in the field and it is universally assumed that the underlying time series are statistically stationary. Are they? Here we ask whether the time series in Fig. 3 are stationary and we use $\langle Q \rangle$ to test the hypothesis. Stationarity implies that $\langle Q \rangle \approx 0$ and the significance of the deviation is judged in units of the standard deviation of steady value $\langle Q \rangle = 0$ via the asymptotics in equation (10) of reference [4]. Calculation of the auto-correlation function for this cosmic ray data shows that it is uncorrelated ("white") so using $\langle Q \rangle$-asymptotics is particularly relevant.

For sake of consistency, we tested a variety of data partitioning, but with the same product $n_t n_T$. Table 1 shows the importance of the approximate $\langle Q \rangle$ algorithm. For n_t approaching unity, dimensions of Q are $\sim 10^4 \times 10^4$ and the $O(N)$ algorithm is crucial. Table 2 shows the calculations. To our surprise, the $\langle Q \rangle$-test consistently detects a presence of a trend beyond reasonable doubt. Specifically, $\langle Q \rangle = 0.06$ gives the confidence limit of 19σ (taking the case $n_t \gtrapprox 100$ for specificity). The associated linear trend is large enough to affect the spectrum and cast doubt on the traditional power-law analysis as the latter implies stationarity via the Wiener–Khintchin theorem.

6. Concluding remarks

In conclusion, we have discovered an $O(N^2)$ calculation of a previously $O(N^4)$ matrix transform with applications in trend detection from noisy data. This increases the efficiency of the transform, and allows access to previously out-of-reach data sample lengths N. For the special case of a small number of samples n_t, we present also an $O(N)$ calculation of trend detection metric $\langle Q \rangle$ which bypasses the need to carry out the full Q-transform. Open access computer codes are provided for both of these calculations.

Table 1
To investigate the range of possibilities, we compare several partitions of the time-series of cosmic rays. By trimming the data from 49223 down to 49140 datapoints, the number of distinct integers n_t, n_T such that $n_t n_T$ equals the total number of kept data points is maximized, providing many partitions for study. The product $n_t n_T$ is held constant. In this table, timing dependence of partitions is shown. For all entries shown, $n_t = 49140/n_T$ is integer. One can see that the fast approximate calculation for $\langle Q \rangle$ is possible for all possible partitions, while those based on full calculation of Q become memory limited past about $n_T = 10^4$, as indicated by the dashes. The approximate calculation stores neither P nor Q, and is able to be performed up to the maximum n_T. Thus, the approximate method is the only way to probe these partitions. For n_T greater than a few hundred, the approximate calculation, consisting of approximating basis vectors and coefficients for m, followed by a Hadamard product with P, is much faster than direct evaluation of Q and its mean element.

n_T	Time Q (s)	Time Q sum (s)	Time m Basis and Coefficients (s)	Time $m_{i,j}P_{i,j}$ sum (s)
49 140	–	–	0.968	1.58E-02
24 570	–	–	0.484	1.15E-02
9828	31.8	8.70E-02	0.200	1.14E-02
4095	2.85	1.53E-02	8.39E-02	7.37E-03
468	1.91E-02	3.95E-04	1.05E-02	7.02E-03
91	5.50E-04	2.94E-05	2.73E-03	7.69E-03
12	4.99E-05	2.34E-05	1.22E-03	7.11E-03

Table 2
Companion to Table 1 for calculated values of $\langle Q \rangle$ under different data partitions. Both exact and approximate calculations give results that are roughly independent of data partition for sufficiently large $n_T \gtrapprox 10^2$. For all entries, $n_t = 49140/n_T$. Accuracy of the approximate calculation is $\delta = \langle Q \rangle_{approximate} / \langle Q \rangle_{exact} - 1$, and is excellent, being better than 10^{-6} where n_T is small enough that the exact calculation can be performed and compared to. $\langle Q \rangle$, normalized by its standard deviation from zero for a comparable white noise process, is about 18, indicating the presence of a trend in the data.

n_T	$\langle Q \rangle_{exact}$	$\langle Q \rangle_{approximate}$	δ	$\langle Q \rangle / \sigma_{white\ noise}$
49 140	–	6.09E-02	–	19.0
24 570	–	6.09E-02	–	19.0
9828	6.10E-02	6.10E-02	3.65E-06	19.0
4095	6.09E-02	6.09E-02	2.13E-06	19.0
468	6.12E-02	6.12E-02	−2.92E-06	19.1
91	6.22E-02	6.22E-02	−7.95E-07	19.1
12	7.27E-02	7.27E-02	−1.78E-15	18.7

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

This work was supported by the National Science Foundation grant AGS-1639868.

Appendix. Mathematical identities for matrix m used in software

It follows from Eq. (17) that $m_{j,k} = m_{k,j}$, a symmetric matrix, and $m_{j,nT-k+1} = -m_{j,k}$, a matrix odd under vertical or horizontal inversion. For the upper left matrix quadrant $j \leq n_T - j$ and $k \leq n_T - k$ it can be shown from (17) that the following is necessary:

$$m_{j,k} = \frac{n_T}{n_t(n_T-1)^2} \times \left(2 \sum_{m=\lfloor \frac{n_T}{2} \rfloor +1}^{n_T-j} \frac{\psi_0(k - \frac{n_T(n_T-m)}{n_T-2m}) - \psi_0(1-k-\frac{mn_T}{n_T-2m})}{n_T-2m} + \frac{2}{n_T}(n_T-2k+1)\left|cos\left(\frac{n_T\pi}{2}\right)\right|\right) \tag{A.1}$$

Here, ψ_0 is the polygamma function of order 0 (e.g. MATLAB psi function, Julia module SpecialFunctions' polygamma function with zero as the first argument). From this, the following can be shown:

$$m_{j+1,k} = m_{j,k} - \frac{2n_T}{n_t(n_T-1)^2} \frac{\psi_0(k - \frac{n_T(n_T-j)}{n_T-2j}) - \psi_0(1-k-\frac{jn_T}{n_T-2j})}{n_T-2j} \tag{A.2}$$

This allows recursion down the columns of m in an exact calculation. Let the subtracted quantity from $m_{j,k}$ be $f(j,k)$. The following is again necessary:

$$m_{j,j} = m_{j+1,j+1} + f(j,j) + f(j,j+1) \tag{A.3}$$

This permits recurrence along the diagonal of m in an exact calculation. Finally, it also is necessary that m satisfies the following:

$$m_{j,k} + m_{j-1,k-1} - m_{j,k-1} - m_{j-1,k} = 2(d^{(1)}_{j-1,k-1} + d^{(2)}_{j-1,k-1}) \tag{A.4}$$

Here $d^{(1)}_{j,k}$ and $d^{(2)}_{j,k}$ are given in Eq. (18) of the main text. This causes m as a whole to be calculable in $\mathcal{O}(n_T^2)$ operations, in contrast to $\mathcal{O}(n_T^4)$ from Eq. (17) ($\mathcal{O}(n_T^2)$ per element).

References

[1] All calculations were performed with the first author's personal Dell Inspiron 15 laptop, equipped with 2.2GHz AMD quadcore processor, using 64 bit Windows 10. The best time scalings were provided by the slower 32 bit version of MATLAB, and thus this is what is shown. For Tables 1 and 2, 64 bit MATLAB was used, being the faster version.

[2] The Pierre Auger Collaboration, Pierre Auger Observatory public data, the public event explorer, 2019, http://labdpr.cab.cnea.gov.ar/ED-en/index.php (ASCII file of all available events downloaded April 28th, 2019).

[3] J. Chang, J. Sha, IEICE Electron. Express 14 (1) (2017) 20161134, http://dx.doi.org/10.1587/elex.13.20161134.

[4] G. Ierley, A. Kostinski, Phys. Rev. X 9 (2019) 031039, http://dx.doi.org/10.1103/PhysRevX.9.031039.

[5] L.T. Nguyen, J. Kim, B. Shim, IEEE Access 7 (2019) 94215–94237, http://dx.doi.org/10.1109/ACCESS.2019.2928130.