Statistics on Data Streams

T. H. Merrett*

McGill University

March 26, 2008

[1]

This is taken from [SZ04, Chap. 5]. It is the first of three applications they develop based on their review of timeseries techniques in the first four chapters.

1. Data streams are unending sequences of data arriving rapidly and needing to be processed. For example, quotations and transactions arriving every second for fifty thousand stocks would constitute a data stream of fifty thousand sequences. Or the Space Shuttle telemeters readings each second for twenty thousand sensors to Houston.

The kind of analyses discussed in chapter 5 of [SZ04] include single-stream statistics such as average, standard deviation and best-fit slope, pairwise statistics such as correlation and "beta", and lagged statistics such as autocorrelation for single streams or cross correlation for pairs of streams.

We should start with a review.

We write an arbitrary timeseries with capital letters, $\mathbf{X}$ or $X_1, X_2, .., X_n$.

Often we *normalize* the timeseries so that its average is 0 and its standard deviation is the the square root of its length, $\sqrt{n}$. We write a normalized series with small letters, $\mathbf{x}$ or $x_1, x_2, .., x_n$.

$$x_i = \frac{X_i - \text{avg}(\mathbf{X})}{\text{std}(\mathbf{X})}$$

where

$$\text{avg}(\mathbf{X}) = \sum_{i=1}^{n} X_i/n$$

and

$$(\text{std}(\mathbf{X}))^2 = \sum_{i=1}^{n} (X_i - \text{avg}(\mathbf{X}))^2/n$$

The *Pearson Correlation Coefficient* between series $\mathbf{X}$ and $\mathbf{Y}$ is

$$\text{corr}(\mathbf{X}, \mathbf{Y}) = \frac{\text{avg}(\mathbf{X}. * \mathbf{Y}) - \text{avg}(\mathbf{X})\text{avg}(\mathbf{Y})}{\text{std}(\mathbf{X})\text{std}(\mathbf{Y})}$$

The author gratefully acknowledges support from the taxpayers of Québec and of Canada who have paid his salary and research grants while this work was developed at McGill University, and from his students (who built the implementations and investigated the data structures and algorithms) and their funding agencies.

where we use the MATLAB symbol for the product of two series (vectors),

$$(\mathbf{X}. * \mathbf{Y})_i = X_i Y_i$$

For a normalized series, the Pearson Correlation Coefficient is just the inner ("dot") product divided by the length

$$\text{corr}(\mathbf{x}, \mathbf{y}) = (\mathbf{x} * \mathbf{y})/n$$

where we use the MATLAB symbol for the inner product of two series (vectors),

$$\mathbf{x} * \mathbf{y} = \sum_{i=1}^{n} x_i y_i$$

The distance between two series is just the Euclidean distance between two points with the terms of the series as coordinates.

$$D^2(\mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{n} (X_i - Y_i)^2$$

The Pearson Correlation Coefficient of two arbitrary series is negatively related to the distance between the normalized series [SZ04, p.92].

$$D^2(\mathbf{x}, \mathbf{y}) = 2n(1 - \text{corr}(\mathbf{X}, \mathbf{Y}))$$

"Beta" between two series gives the relative movement of the first to the second

$$\text{beta}(\mathbf{X}, \mathbf{Y}) = \frac{\sum_{i=1}^{n} (X_i - \text{avg}(\mathbf{X}))(Y_i - \text{avg}(\mathbf{Y}))}{\sum_{i=1}^{n} (Y_i - \text{avg}(\mathbf{Y}))^2}$$

With stocks, $\mathbf{X}$ is often compared with the market, $\mathbf{Y}$, to assess the "risk" of $\mathbf{X}$. Try it with $\mathbf{X} =$ 2,2,$-2, -2$ and $\mathbf{Y} =$ 1,1,$-1, -1$.

2. Statistics such as correlation and beta are based on the inner product between two series, so [SZ04] focus on speeding up calculating the inner product.

They do this by approximation. First, the inner product of the Fourier transforms of two series equals the inner product of the series. This is obvious because the Fourier transform is an orthonormal linear transform, like the matrix that changes the axes of a vector space. The inner product of two vectors in this space is unchanged by axis transformation, and so is the same in the new coordinate system, e.g. the Fourier-transformed series.

Second, the Fourier transform can be approximated by omitting many small coefficients (in many cases but not all). We've discussed this for multimedia data compression. The inner product of the approximations is always smaller than the inner product of the full Fourier-transformed series, but we can still use it to estimate correlation, or distance, between two series.

The relationships between correlation and the approximate distance given by the selected coefficients of the Fourier transform are [SZ04, pp.112–3]

$$\text{corr}(\mathbf{X}, \mathbf{Y}) \geq 1 - \epsilon^2 \Rightarrow D_{\text{approx}}(\mathbf{xF}, \mathbf{yF}) \leq \sqrt{n}\epsilon$$

$$\text{corr}(\mathbf{X}, \mathbf{Y}) \leq -1 + \epsilon^2 \Rightarrow D_{\text{approx}}(-\mathbf{xF}, \mathbf{yF}) \leq \sqrt{n}\epsilon$$

where **xF** is our notation for the Fourier transform of the normalized series, **x**, and similarly **yF** and so on. (We'll write **XF** to specify the Fourier transform of a non-normalized series.)

Here, ϵ^2 is a small threshold. So to ensure that corr(**X**,**Y**) is within 1/4 of 1 (fully correlated), given **X**, we need to find **Y** such that the approximated distance between the Fourier-transformed normalized **x** and **y** is at most $\sqrt{n}/2$.

The second relationship above allows us to find negatively-correlated series.

Note that the approximation to the Fourier-transformed inner product is not monotonic, and does not avoid false negatives. That is, it could miss some highly-correlated series if we used only a correlation threshold. But we are saved by the relationships with the Fourier-transformed distances, above.

3. Fourier-transforming an entire series would be expensive, especially since, if it is a data stream, the series is non-ending. So Shasha and Zhu work with sliding windows of, say, an hour's worth of (3600) points or so.

In addition to the sliding windows, for which the statistics are to be gathered, Shasha and Zhu use a smaller unit, of say a couple of minutes (120 points) or so, as an intermediate. They call these "basic windows" but we'll call them *blocks*.

A block has b points, a sliding window has w points, and we'll say that an integer number, k, of blocks make up a window, $w = bk$.

The idea is to Fourier-transform the block, once all its points have been received, and to store only "digests": f selected Fourier coefficients and two running sums (of the points and of their squares) instead of the data points themselves for the block. This is $2 + 2f$ numbers, since the Fourier coefficients are complex.

The sum, sum-of-squares and inner product for the sliding window can be computed on the fly from these digests for each block. If s_j is the sum of points for block $j = c, .., c+k-1$ for a window commencing at block c, q_j is the sum of squares, and d_j is the inner product of a pair of series, then as the window moves from blocks $c, .., c+k-1$ to blocks $c+1, .., c+k$ we can replace the corresponding quantities for the window.

$$\begin{aligned} S_{c+1} &= S_{c+k-1} + s_{c+k} - s_c \\ Q_{c+1} &= Q_{c+k-1} + q_{c+k} - q_c \\ D_{c+1} &= D_{c+k-1} + d_{c+k} - d_c \end{aligned}$$

Of course we would have to store d for all pairs of series in the data stream, $N!2$ (N choose 2) different numbers. So it is better to construct any d needed from the stored Fourier coefficients. we do not repeat the formula [SZ04, p.114], which is in the same spirit as the three calculations above.

4. It is time for an example. We simplify rather heavily from, say, 24 300-point blocks per sliding window. Let's look at 16-point blocks, $b = 16$.

Here are four input series for a single block, written as column vectors. We'll call them **w**, **x**, **y** and **z**, respectively.

```
 1    1   -1   -1
 1    1   -1   -1
 1   -1    1   -1
-1   -1    1    1
 1    1   -1   -1
 1    1   -1   -1
-1   -1    1    1
```

```
    -1    -1     1     1
     1     1    -1    -1
     1     1    -1    -1
    -1    -1     1     1
    -1    -1     1     1
     1     1    -1    -1
    -1     1    -1     1
    -1    -1     1     1
    -1    -1     1     1
```

Their inner products are

```
    16    12   -12   -16
    12    16   -16   -12
   -12   -16    16    12
   -16   -12    12    16
```

so, since they are all normalized, their correlations are

```
    1     3/4  -3/4  -1
    3/4   1    -1    -3/4
   -3/4  -1     1     3/4
   -1    -3/4   3/4   1
```

MATLAB calculates the Fourier matrix using `makeDFT(16)`:

```
%makeDFT.m THM 060704
%function Fn = makeDFT(n) generates n*n Fourier transform matrix
function Fn = makeDFT(n)
for j=1:n, for k=1:n, En(j,k)=mod((j-1)*(k-1),n); end; end;
Fn = exp((i*2*pi/n).*En)./sqrt(n);
```

Applied to the four series gives the fourier-transformed series.

```
       0                  0                  0                  0
   0.1622 + 0.8155i   0.0000 + 0.0000i  -0.0000 - 0.0000i  -0.1622 - 0.8155i
   0.3536 + 0.8536i   0.0000 + 0.0000i  -0.0000 - 0.0000i  -0.3536 - 0.8536i
   0.1084 + 0.1622i  -0.0000 - 0.0000i   0.0000 + 0.0000i  -0.1084 - 0.1622i
   1.5000 + 1.5000i   2.0000 + 2.0000i  -2.0000 - 2.0000i  -1.5000 - 1.5000i
  -0.8155 - 0.5449i   0.0000 + 0.0000i  -0.0000 - 0.0000i   0.8155 + 0.5449i
  -0.3536 - 0.1464i         0                  0             0.3536 + 0.1464i
   0.5449 + 0.1084i  -0.0000 - 0.0000i   0.0000 + 0.0000i  -0.5449 - 0.1084i
   1.0000 - 0.0000i         0                  0            -1.0000 + 0.0000i
   0.5449 - 0.1084i   0.0000 + 0.0000i  -0.0000 - 0.0000i  -0.5449 + 0.1084i
  -0.3536 + 0.1464i   0.0000 + 0.0000i  -0.0000 - 0.0000i   0.3536 - 0.1464i
  -0.8155 + 0.5449i  -0.0000 - 0.0000i   0.0000 + 0.0000i   0.8155 - 0.5449i
   1.5000 - 1.5000i   2.0000 - 2.0000i  -2.0000 + 2.0000i  -1.5000 + 1.5000i
   0.1084 - 0.1622i   0.0000 + 0.0000i  -0.0000 - 0.0000i  -0.1084 + 0.1622i
   0.3536 - 0.8536i   0.0000            -0.0000            -0.3536 + 0.8536i
   0.1622 - 0.8155i  -0.0000 - 0.0000i   0.0000 + 0.0000i  -0.1622 + 0.8155i
```

Note that coefficients 15 are the complex conjugates of coefficients 1 (counting rows above from 0) and so on for pairs (14,2), (13,3), etc. This is generally true for Fourier coefficents of a series of real values, and is one of the symmetry properties of Fourier transforms [SZ04, p.20].

To help us select coefficients (Shasha and Zhu just take the first f, but in this simple example we'll take the largest ones, in order to keep the number of digests to a minimum), we look at the magnitudes of all these coefficients.

```
     0         0         0         0
0.6913    0.0000    0.0000    0.6913
0.8536    0.0000    0.0000    0.8536
0.0381    0.0000    0.0000    0.0381
4.5000    8.0000    8.0000    4.5000
0.9619    0.0000    0.0000    0.9619
0.1464         0         0    0.1464
0.3087    0.0000    0.0000    0.3087
1.0000         0         0    1.0000
0.3087    0.0000    0.0000    0.3087
0.1464    0.0000    0.0000    0.1464
0.9619    0.0000    0.0000    0.9619
4.5000    8.0000    8.0000    4.5000
0.0381    0.0000    0.0000    0.0381
0.8536    0.0000    0.0000    0.8536
0.6913    0.0000    0.0000    0.6913
```

From this we see that coefficients 4 and 12 (counting from row 0) dominate all four series.

We can confirm that the inner products of the Fourier-transformed series are the same as the inner products of the original series.

```
 16.0000             12.0000 + 0.0000i -12.0000 - 0.0000i -16.0000
 12.0000 - 0.0000i   16.0000           -16.0000           -12.0000 + 0.0000i
-12.0000 + 0.0000i -16.0000            16.0000            12.0000 - 0.0000i
-16.0000           -12.0000 - 0.0000i  12.0000 + 0.0000i  16.0000
```

Now we approximate by setting all coefficients to zero except for 4 and 12. The approximate inner products are

```
  9   12  -12   -9
 12   16  -16  -12
-12  -16   16   12
 -9  -12   12    9
```

Note that we do not need to keep coefficients 12, since they are the complex conjugates of coefficients 4. In fact, the inner product values can be found by calculating inner products with only coefficients 4 non-zero, and then doubling.

It is not surprising that we got perfect results for inner products involving series $\mathbf{x}$ and $\mathbf{y}$, since these are both periodic and the coefficients 4 and 12 are the only nonzero Fourier coefficients: the approximation is exact. Here are the series transformed back again from the approximate Fourier transform.

```
  0.7500              1.0000              -1.0000             -0.7500
  0.7500 + 0.0000i    1.0000 + 0.0000i   -1.0000 - 0.0000i   -0.7500 - 0.0000i
 -0.7500 - 0.0000i   -1.0000 - 0.0000i    1.0000 + 0.0000i    0.7500 + 0.0000i
 -0.7500 - 0.0000i   -1.0000 - 0.0000i    1.0000 + 0.0000i    0.7500 + 0.0000i
  0.7500              1.0000              -1.0000             -0.7500
  0.7500 + 0.0000i    1.0000 + 0.0000i   -1.0000 - 0.0000i   -0.7500 - 0.0000i
 -0.7500 - 0.0000i   -1.0000 - 0.0000i    1.0000 + 0.0000i    0.7500 + 0.0000i
 -0.7500 - 0.0000i   -1.0000 - 0.0000i    1.0000 + 0.0000i    0.7500 + 0.0000i
  0.7500              1.0000              -1.0000             -0.7500
  0.7500 + 0.0000i    1.0000 + 0.0000i   -1.0000 - 0.0000i   -0.7500 - 0.0000i
 -0.7500 - 0.0000i   -1.0000 - 0.0000i    1.0000 + 0.0000i    0.7500 + 0.0000i
 -0.7500 - 0.0000i   -1.0000 - 0.0000i    1.0000 + 0.0000i    0.7500 + 0.0000i
  0.7500              1.0000              -1.0000             -0.7500
  0.7500 + 0.0000i    1.0000 + 0.0000i   -1.0000 - 0.0000i   -0.7500 - 0.0000i
 -0.7500 - 0.0000i   -1.0000 - 0.0000i    1.0000 + 0.0000i    0.7500 + 0.0000i
 -0.7500 - 0.0000i   -1.0000 - 0.0000i    1.0000 + 0.0000i    0.7500 + 0.0000i
```

Clearly, **x** and **y** are identical to the original series, and **w** and **z** are periodic variants of the originals, differing in sign in only two places each.

5. Now we can use the relationships of Note 2, above, to find similar (closely correlated) pairs of series, as well as opposite (closely negatively correlated) pairs of series. We do this in the context of the single block we calculated above, rather than for a full sliding window of k blocks. Shasha and Zhu use the full window, as we should do.[2]

First, we can find the true correlations, corr(), between pairs of the original series, and we can find the true distances-squared, $D^2()$, between pairs of the full Fourier-transformed series.

corr()

	w	x	y	z
w	1	3/4	-3/4	-1
x	3/4	1	-1	-3/4
y	-3/4	-1	1	3/4
z	-1	-3/4	3/4	1

distance-squared()

	w	x	y	z
w	0	8	56	64
x	8	0	64	56
y	56	64	0	8
z	64	56	8	0

Confirm that these satisfy the property given in Note 1, above.

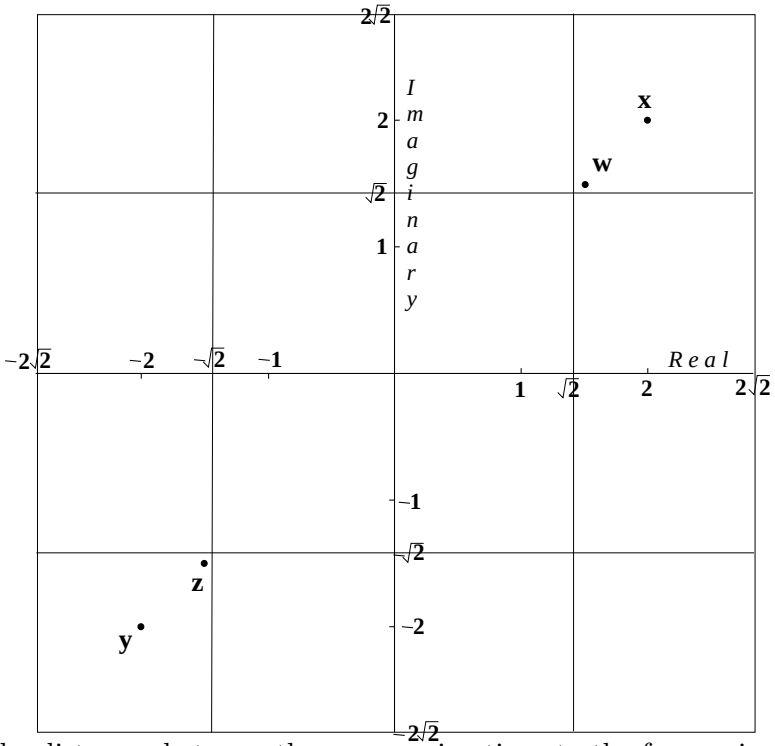
Since we have retained only $f = 1$ Fourier coefficient in the approximation for each series, we can locate each series in a two-dimensional space ($2f$, with different dimensions for real and imaginary parts of the complex coefficients).

These coefficients are

```
        w                   x                   y                   z
  1.5000 + 1.5000i    2.0000 + 2.0000i   -2.0000 - 2.0000i   -1.5000 - 1.5000i
```

Here is the plot.

[2] On p.114, Shasha and Zhu take the Fourier transformation over all w points of the full window, while on p.110 they take the Fourier transformation over the b points of the block. We use the above 16-point Fourier transform, and say that $w = 16$.



We can calculate the distances between these approximations to the four series, or just read them off the plot.

```
distance-squared()
    w   x   y   z
w   0   1   7   6
x   1   0   8   7
y   7   8   0   1
z   6   7   1   0
```

Let's use these approximate distances to find pairs of series that correlate within 1/4, i.e., corr() $\geq 1 - 1/4$. Thus, $\epsilon = 1/2$, so we need approximate distances $\sqrt{w}\epsilon = \sqrt{16}/2 = 2$, i.e., distance-squared ≤ 4. We see this is true for pairs (**w**,**x**) and (**y**,**z**).

We can use the second relationship to find pairs that anticorrelate within 1/4, i.e., corr() ; $e-1+1/4$. Here we need distance-squared of the one series with the *negative* of the other to be ≤ 4, and we see that this maps **w** right on top of **z** and **x** right on top of **y**. So the anticorrelated pairs are (**w**,**z**), (**x**,**y**), (**w**,**y**) and (**x**,**z**).

Shasha and Zhu apply a grid to plots such as the above two-dimensional plot in such a way that we need look only in the grid cell containing one series, or in the immediately adjoining grid cells, to find other series within the prescribed distance. The grid depends on ϵ.

In our example, the grid cell width should be $\sqrt{w}\epsilon = 2$. The whole grid must be able to contain all possible values of the Fourier coefficients. Since the sum of squares of Fourier coefficients of a normalized series equals w, by definition of normalization, the largest any coefficient can be is $\sqrt{w/2}$. (The 2 comes from the fact that coefficients are paired, as we saw above, into two complex conjugates, in the case (always!) that the series is real-valued.)

For our example, $\sqrt{w/2} = \sqrt{8} = 2\sqrt{2}$, and this is the absolute value of the coordinates of the outer box shown in the plot above.

The number of grid cells of width $\sqrt{w}\epsilon$ that can fit into length (of the positive half of either axis)

$\sqrt{w/2}$ is

$$\lceil \frac{\sqrt{w}\epsilon}{\sqrt{w/2}} \rceil = \lceil \frac{1}{\sqrt{2}\epsilon} \rceil = \lceil \frac{2}{\sqrt{2}} \rceil = 2$$

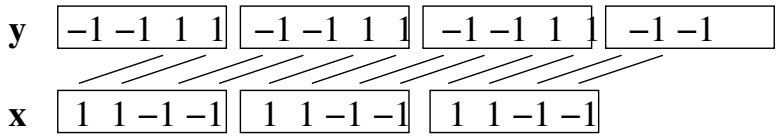
Thus we showed two cells along each half-axis, above, making sixteen cells in all, of width $\sqrt{2}$ instead of 2. (Excursion: can we change ϵ *after* building the grid?)

We still need to search only the cell and its immediate neighbours to find series within $\sqrt{w}\epsilon$ of a given series. As it happens, in our example the closely-correlated series all appear in the same cells as each other.

This grid structure is a specially-tuned special case of *multipaging* [Mer78, Mer99], devised for orthogonal range queries, nearest-neighbour and other multidimensional searches.

6. The correlations discussed so far are without lag: each point in a block for one series is combined with the corresponding point in the block for the other series (or for itself).

Not only are **x** and **y** anticorrelated, but they also are fully correlated with a lag of 2. Let's look at this with an even smaller block, $b = 4$, in the case of windows of three blocks, $k = 3, w = 12$. We designate the lag as $\ell = 2$, and we will need $b - \ell$ (which also equals 2 so we must be a little careful in the following example).



We can write and rearrange the sum giving the inner product to see more clearly what we need to do.

$$\begin{aligned}
\sum_{i=1}^{w} x_i y_{i+\ell} &= \sum_{j=1}^{k} x_{(j-1)b+1} y_{(j-1)b+3} + x_{(j-1)b+2} y_{(j-1)b+4} \\
&\quad + x_{(j-1)b+3} y_{jb+1} + x_{(j-1)b+4} y_{jb+2} \\
&= \sum_{j=1}^{k} \sum_{a=1}^{b-\ell} x_{(j-1)b+a} y_{(j-1)b+a+\ell} + \sum_{a=1}^{\ell} x_{(j-1)b+b-\ell+a} y_{jb+a} \\
&= \sum_{j=1}^{k} \sum_{a=(j-1)b+1}^{jb-\ell} x_a y_{a+\ell} + \sum_{a=jb-\ell+1}^{jb} x_a y_{a+\ell}
\end{aligned}$$

The last two equations are valid for any b and ℓ, not just 4 and 2.

To illustrate how this affects the Fourier transform, we transform the block not the window, following p.110 rather than p.114 of [SZ04]. This enables us to show the Fourier transform matrix, which is 1/2 times

```
1  1  1  1
1  i -1 -i
1 -1  1 -1
1 -i -1  i
```

That is, half of $e^{i\pi/4} = i$ raised to the power of {0,1,2,3}*{0,1,2,3} mod 4:

```
0  0  0  0
0  1  2  3
0  2  0  2
0  3  2  1
```

Since

$$\frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}\begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \end{pmatrix}\begin{pmatrix} -1 \\ -1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1+i \\ 0 \\ 1-i \end{pmatrix}\begin{pmatrix} 0 \\ -(1+i) \\ 0 \\ -(1-i) \end{pmatrix} = \begin{pmatrix} 0 \\ xF_2 \\ 0 \\ xF_4 \end{pmatrix}\begin{pmatrix} 0 \\ yF_2 \\ 0 \\ yF_4 \end{pmatrix}$$

we can find the components of x_i and y_i in terms of the Fourier coefficients xF_i and yF_i by using the inverse Fourier matrix

$$\frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}\begin{pmatrix} 0 \\ xF_2 \\ 0 \\ xF_4 \end{pmatrix}\begin{pmatrix} 0 \\ yF_2 \\ 0 \\ yF_4 \end{pmatrix}$$

This works out, for the two terms in the same block, to

$$x_1y_3^* + x_2y_4^* = -\frac{1}{2}(xF_2yF_4^* + xF_4yF_2^*)$$

and, for the two terms crossing blocks, to

$$x_3y_5^* + x_4y_6^* = x_3y_1^* + x_4y_2^* = -\frac{1}{2}(xF_2yF_4^* + xF_4yF_2^*)$$

($y_5^* = y_1^*$ and $y_6^* = y_2^*$ in our example because the series are periodic.)

(You'll find it informative to work out which elements of the matrix are combined with which other elements to get the -1/2 factors: they are in the second and fourth columns, of course, and some combine to 0, some to -1/2.)

The purpose of this oversimplified example is to show that lag gives rise to cross-terms in the Fourier expansion. Note that xF_2 combines with yF_4 above, and vice versa. For the non-lagged case, xF_i would only combine with yF_i—for the same i—because of the orthogonality of the rows (columns) of the Fourier transform.

Writing the above expressions in matrix form

$$\begin{aligned} x_1y_3^* + x_2y_4^* &= (xF_2, xF_4)\begin{pmatrix} 0 & -1/2 \\ -1/2 & 0 \end{pmatrix}\begin{pmatrix} yF_2^* \\ yF_4^* \end{pmatrix} \\ x_3y_5^* + x_4y_6^* &= (xF_2, xF_4)\begin{pmatrix} 0 & -1/2 \\ -1/2 & 0 \end{pmatrix}\begin{pmatrix} yF_2^* \\ yF_4^* \end{pmatrix} \end{aligned}$$

while the matrices for the unlagged case would be diagonal.

7. Putting all this together, we have, for our example, $f = 1$ Fourier coefficient, i.e., $2f = 2$ numbers, for each block. If we include the block sum-of-terms and sum-of-squares, this is a record (tuple) of $2f + 2$ numbers, say $4(2f + 2)$ bytes.

These statistics are summarized from the b data points of the block at, say, 4 bytes each. In our example, $b = 16$ and

$$4b : 4(2f + 2) = 64 : 16$$

so we save a factor of 4 on even this baby example.

This is for one data stream, so if there are N data streams, the block must also include a stream identifier, say 2 bytes (the U.S. trades 50,000 stocks; NASA monitors 20,000 sensors on the space

shuttle): $N(4(2f+2)+2)$ bytes for N tuples.

Since a sliding window has k blocks, we must retain k times this many tuples, and add a block identifier of, say, 1 byte: $kN(4(2f+2)+3)$ bytes for kN tuples. The block identifier is the sequence number on which the timeseries is ordered.

In addition, we have the statistics we are compiling from these block digests for the entire sliding window. But these are only a handful of numbers for the whole window.

This counting is correct both for our examples in which we Fourier-transformed only within each block and for the proper way of doing it in which we must Fourier-transform the entire sliding window.

We can use **redwin(k)** to pick out the window. But with data streams, which have indefinite length, in practice the stored data will be only the window, replacing the first block by the next block we come to as we move on.

If we have privacy mechanisms, which prevent certain users from seeing certain data, a data stream can be thought of as a relation with global privacy outside the current sliding window: nobody can see data in the future of the latest block of the window or in the past of the first block of the window.

Here are the numbers and the relational representation of an example from [SZ04, p.115]. Sixteen Fourier coefficients are kept for blocks of $b = 300$ (5 minutes) timepoints and used to build statistics for windows of $w = 7200$ (2 hours) timepoints: $k = 24$, $f = 16$, $2f + 2 = 34$, $4(2f + 2) + 3 = 139$ bytes per tuple.

The storage requirements are, for N data streams

N	1000	10,000	100,000
B	139K	1.39M	13.9M
W	3.34M	33.4M	334M

where B is the number of bytes in a block and W is the number of bytes in a window. (Note that 2 bytes cannot identify each of 100,000 streams, so we'd have to add another couple of bytes per record.)

The relation is

Window
(*seq stream S Q f01r f01i f02r f02i f03r f03i f04r f04i*
f05r f05i f06r f06i f07r f07i f08r f08i f09r f09i f10r f10i
f11r f11i f12r f12i f13r f13i f14r f14i f15r f15i f16r f16i)

and has 24 different sequence numbers.

References

[Mer78] T. H. Merrett. Multidimensional paging for efficient data base querying. In *International Conference on Data Base Management Systems*, pages 277–90, Milano, June 1978.

[Mer84] T. H. Merrett. *Relational Information Systems*. Reston Publishing Co., Reston, VA., 1984.

[Mer99] T. H. Merrett. Relational information systems. (revisions of [Mer84]):
Data structures for secondary storage: http://www.cs.mcgill.ca/∼cs420
Database programming: http://www.cs.mcgill.ca/∼cs612, 1999.

[SZ04] Dennis Shasha and Yunyue Zhu. *High Performance Discovery in Time Series: Techniques and Case Studies*. Springer Verlag, New York, 2004.