KDD2017

Coffee Break

© Biblioteca Nacional de España

Reconvene

- In the first half, we have seen many wonderful things you can do with the Matrix Profile, *without* explaining how to compute it!
- In this half, we will describe algorithms to compute matrix profile, optimization techniques for scalability, portability to modern hardware, approximation to gain speed, and extension to special cases.
- We embed MATLAB scripts in slides that can reproduce charts and numbers, and explain algorithms.
- Slides are text heavy to facilitate offline readers. In the presentation, focus is on figures.
- Let's begin…

Outline

Act 1

- Our Fundamental Assumption
- What is the (MP) Matrix Profile?
- Properties of the MP
- Developing a Visual Intuition for MP
- Basic Algorithms
  - MP Motif Discovery
  - MP Time Series Chains
  - MP Anomaly Discovery
  - MP Joins (self and AB)
  - MP Semantic Segmentation
- From Domain Agnostic to Domain Aware: The Annotation Vector (A simple way to use domain knowledge to adjust your results)
- The "*Matrix Profile and ten lines of code is all you need*" philosophy.
- Break

Act 2

- Background on time series mining
  - Similarity Measures
  - Normalization
- Distance Profile
  - Brute Force Approach
  - Just-in-time Normalization
  - The MASS Algorithm
  - Extensions of MASS
- Matrix Profile
  - STAMP
  - STOMP
  - GPU-STOMP
  - SCRIMP
- Open problems to solve

What are Time Series? 1 of 2

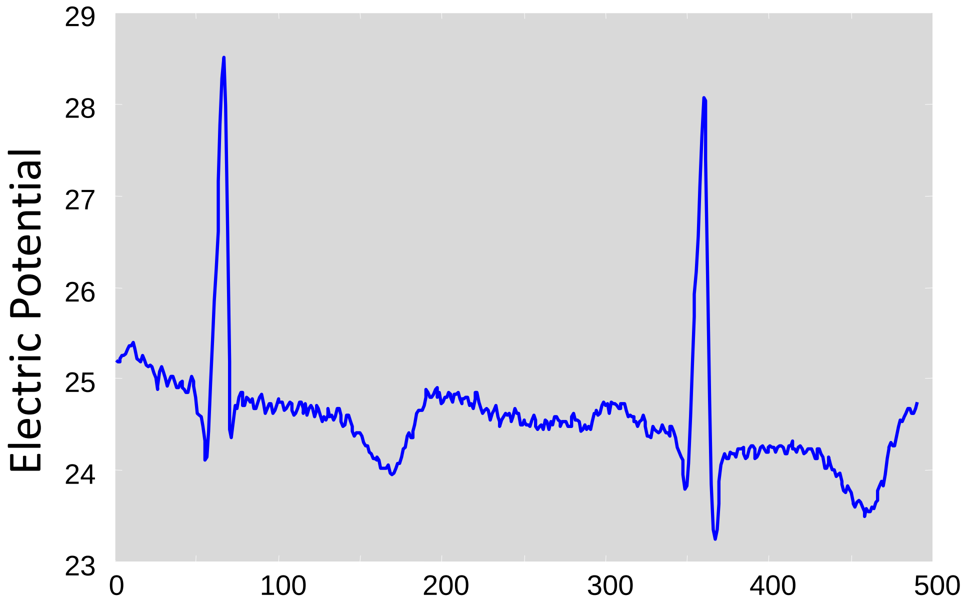
A time series is a collection of observations made sequentially in time.

More than most types of data, time series lend themselves to *visual* inspection and intuitions…

For example, looking at the numbers in this blue vector tells us nothing.
But after *plotting* the data, we can recognize a heartbeat, and possibly even diagnose this person's disease.

When the observations are uniformly sampled, the index of observation can replace the time of observation. In the rest of the tutorial, we assume time series are vectors.

The observations may have a unit.

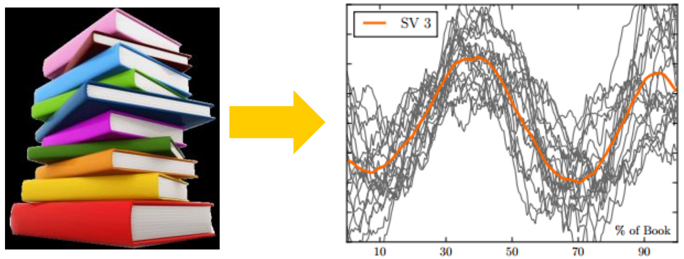


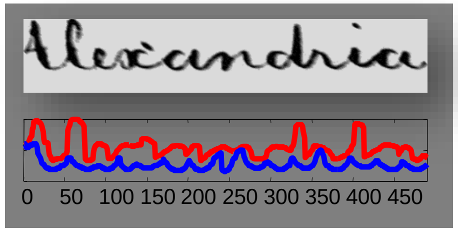
25.350
25.350
25.400
25.400
25.325
25.225
25.200
25.175
..
24.625
24.675
24.675
24.675

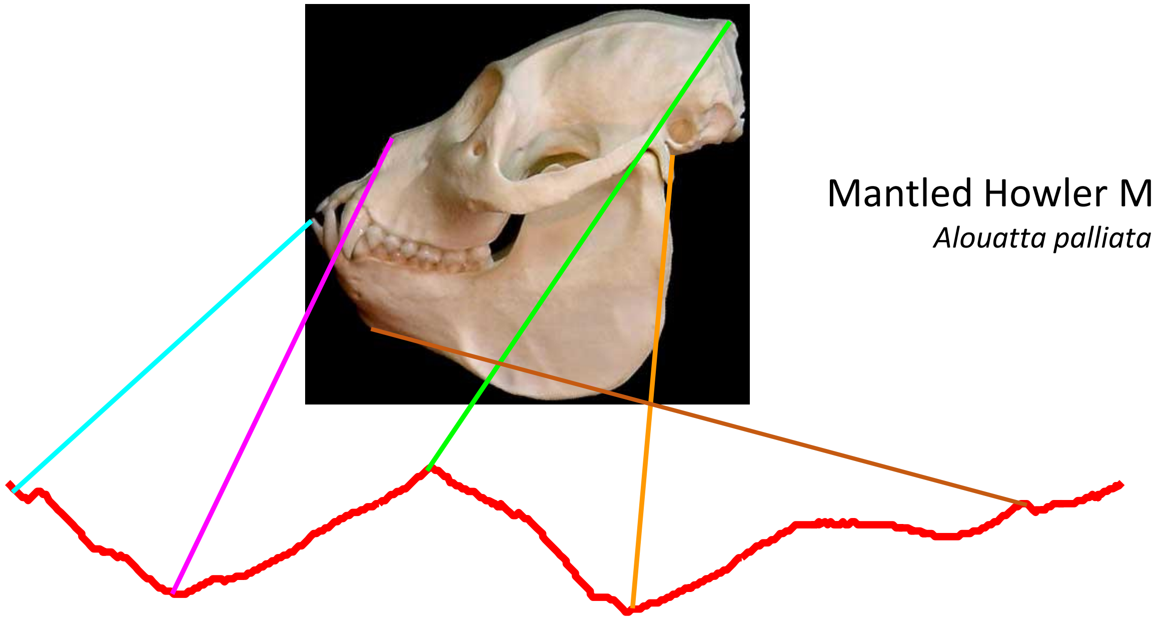
What are Time Series?   2 of 2

As an aside… (not the main point for today)

Many types of data that are not *true* time series can be fruitfully transformed into time series, including DNA, speech, textures, core samples, ASCII text, historical handwriting, novels and even *shapes*.







Mantled Howler Monkey
Alouatta palliata

Similarity Measures for Time Series

- A similarity measure compares two time series and produces a number representing their similarity
  - A distance measure is the opposite of similarity measure
- Lockstep Measures
  - Euclidean Distance
  - Correlation Coefficient
  - Cosine Similarity
- Elastic Measures
  - Dynamic Time Warping
  - Edit Distance
  - Longest Common Subsequence

Euclidean Distance Metric

Given two time series

$\mathbf{x} = x_1 \ldots x_n$

and

$\mathbf{y} = y_1 \ldots y_n$

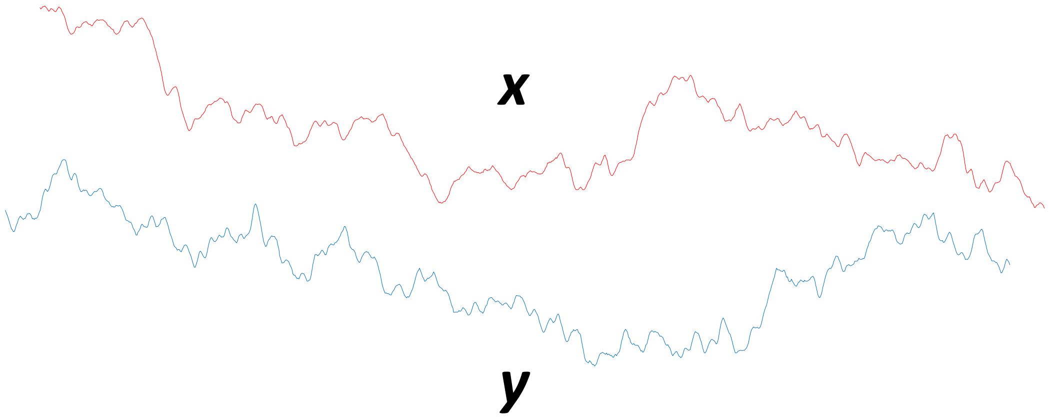
their z-Normalized Euclidean distance is defined as:

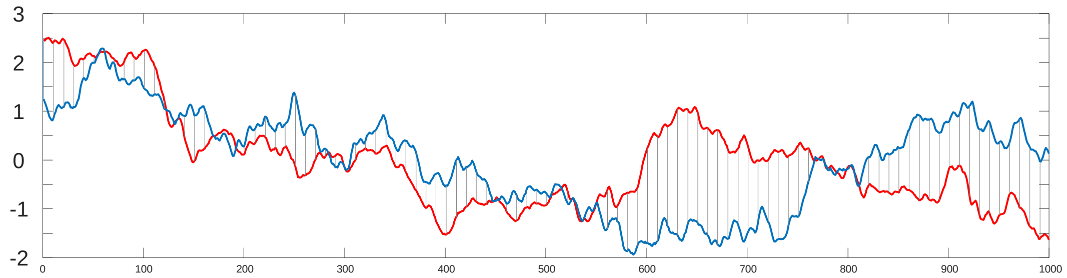
$$\widehat{x_i} = \frac{x_i - \mu_x}{\sigma_x} \qquad \widehat{y_i} = \frac{y_i - \mu_y}{\sigma_y}$$

```
function y = zNorm(x)
y = (x-mean(x))/std(x,1);
```

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (\widehat{x_i} - \widehat{y_i})^2}$$

```
function d = EuclideanDistance(x,y)
d = sqrt(sum((x-y).^2));
```





Pearson’s Correlation Coefficient

- Given two time series $\boldsymbol{x}$ and $\boldsymbol{y}$ of length m.
- Correlation Coefficient:

$$corr(\boldsymbol{x},\boldsymbol{y}) = \frac{(E(x)-\mu_x)(E(y)-\mu_y)}{\sigma_x\sigma_y} = \frac{\sum_{i=1}^{m} x_i y_i - m\mu_x\mu_y}{m\sigma_x\sigma_y}$$

  - Where $\mu_x = \frac{\sum_{i=1}^{m} x_i}{m}$ and $\sigma_x^2 = \frac{\sum_{i=1}^{m} x_i^2}{m} - \mu_x^2$

- Sufficient Statistics:

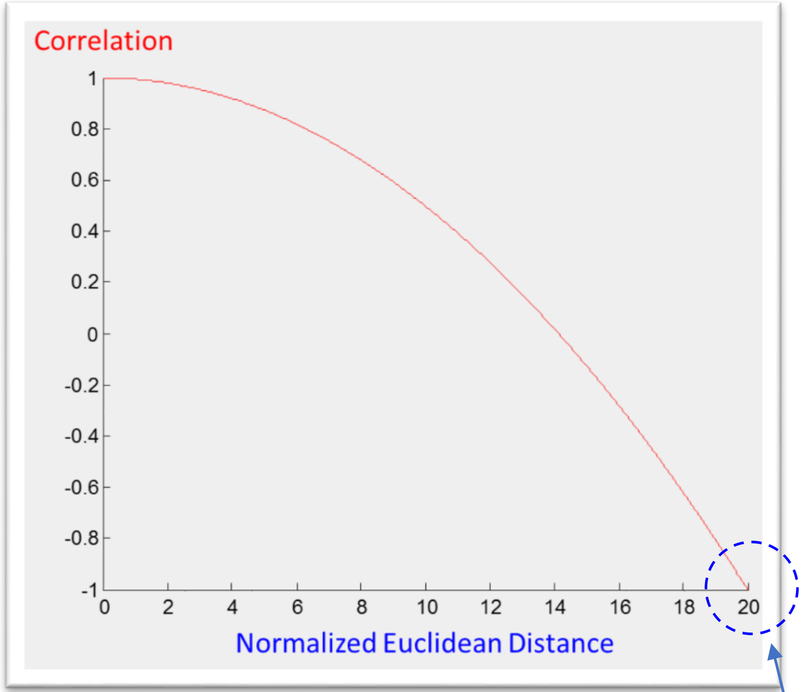
$$\sum_{i=1}^{m} x_i y_i \quad \sum_{i=1}^{m} x_i \quad \sum_{i=1}^{m} y_i \quad \sum_{i=1}^{m} x_i^2 \quad \sum_{i=1}^{m} y_i^2$$

The sufficient statistics can be calculated in one linear scan. Given the sufficient statistics, correlation coefficient is a constant operation. Note the use of the dot product, which is the key component of many lockstep measures.

Relationship with Euclidean Distance

$$d(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) = \sqrt{2m(1 - corr(\boldsymbol{x}, \boldsymbol{y}))}$$

- Correlation coefficient does not obey triangular inequality, while Euclidean distance does
- Maximizing correlation coefficient can be achieved by minimizing normalized Euclidean distance and vice versa
- Correlation coefficient is bounded between -1 and 1, while z-normalized Euclidean distance is bounded between zero and a positive number dependent on m



20 for m = 100

$$d^2(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) = \sqrt{2m(1 - corr(\boldsymbol{x}, \boldsymbol{y}))}$$

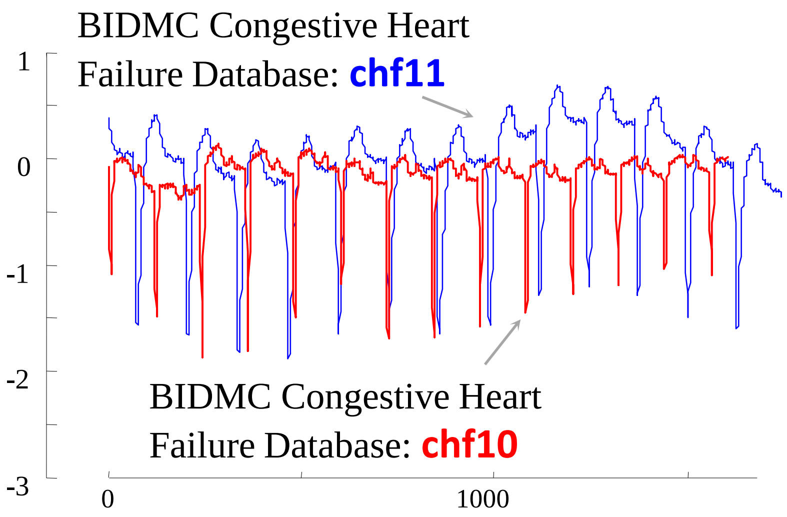
Abdullah Mueen, Suman Nath, Jie Liu: Fast approximate correlation for massive time-series data. SIGMOD Conference 2010: 171-182

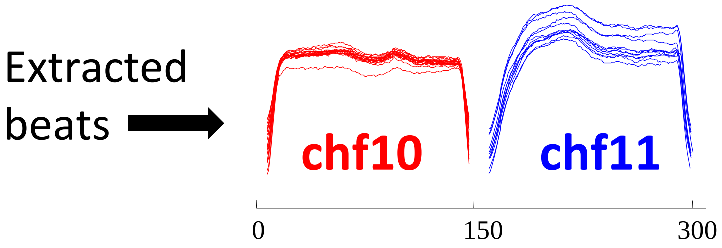
Working Formula

$$d(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) = \sqrt{2m(1 - \frac{\sum_{i=1}^{m} x_i y_i - m\mu_x\mu_y}{m\sigma_x\sigma_y})}$$

- We will use the above z-Normalized Euclidean distance as the similarity measure for the rest of the presentation
- We claim calculating Matrix Profile for Correlation Coefficient and Cosine Similarity is trivial given an algorithm for z-Normalized Euclidean distance

The Importance of z-Normalization and correlation 1 of 2





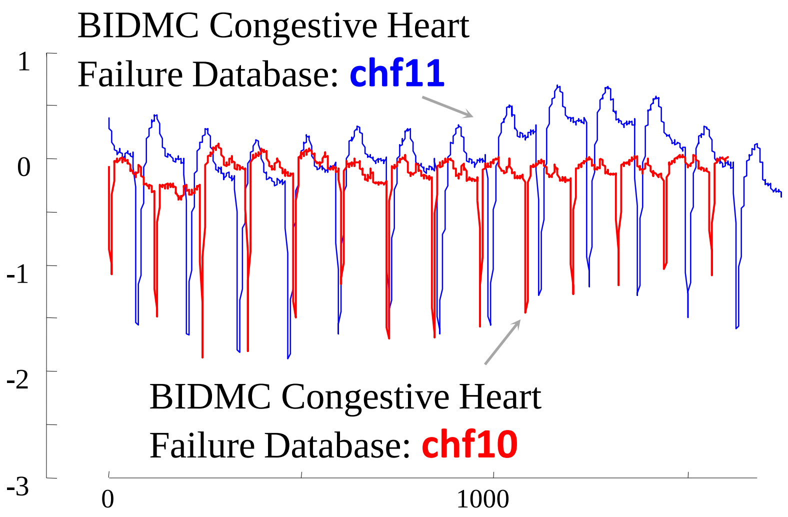
Essentially all datasets must have *every* subsequence z-normalized.

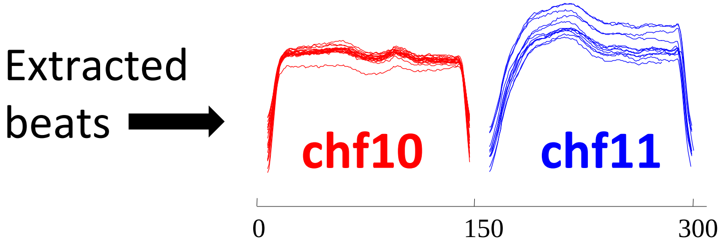
There are a handful of occasions where it does not make sense to z-normalize, but in those cases, similarity search does not make sense either.

In this example, we begin by extracting heartbeats from two unrelated people.

Even without normalization, it happens that both sets have almost the same mean and standard deviation. Given that, do we need to bother to normalize them? (next slide)

The Importance of z-Normalization and correlation 2 of 2

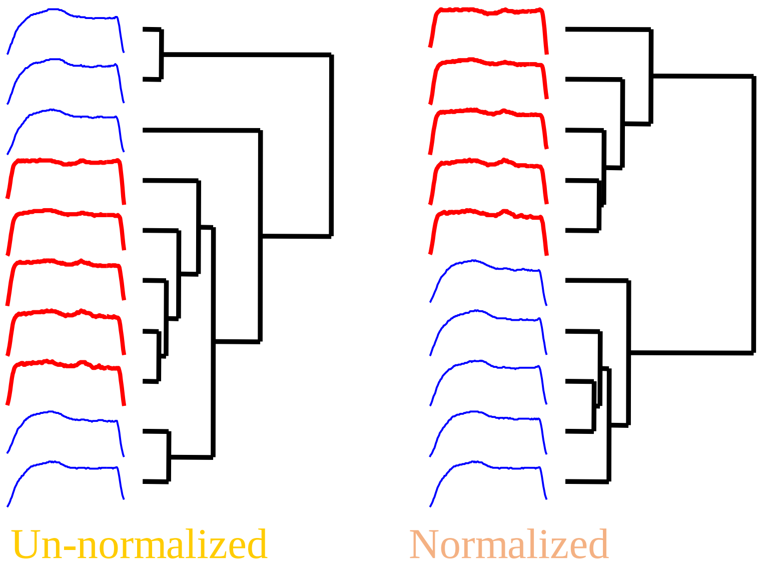




Surprisingly z-normalizing can be a computational bottleneck, but later we will show you how to fix that.

Without normalization, the results are *very* poor, some blue heartbeats are closer to red heartbeats than there are to another blue beat .

With normalization, the results are perfect.



In this example, we extracted heartbeats from two different time series, and clustered them with and without normalization.

Outline

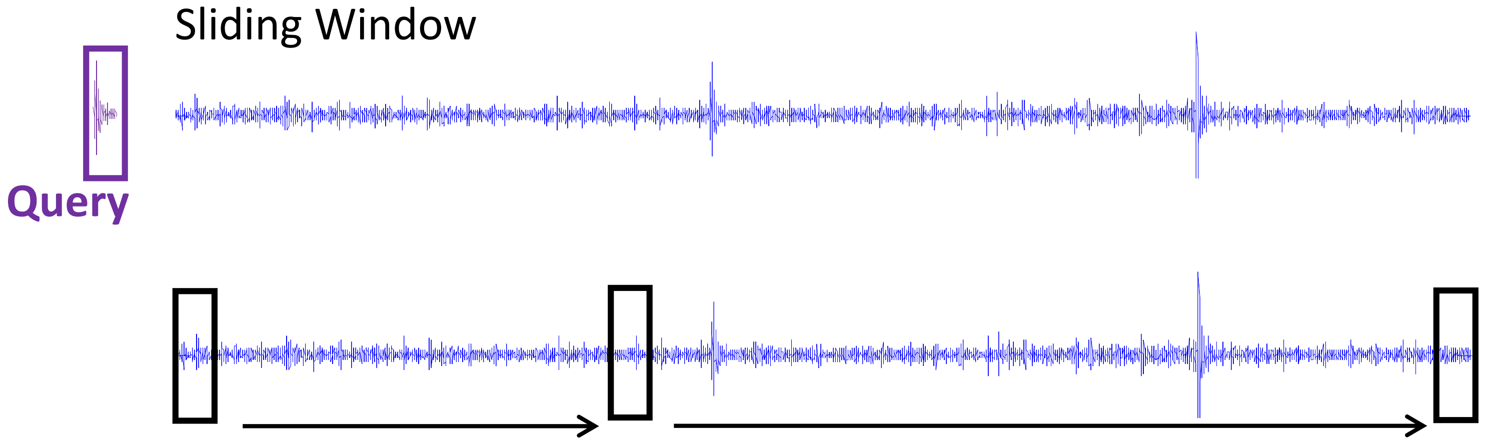
Act 1

- Our Fundamental Assumption
- What is the (MP) Matrix Profile?
- Properties of the MP
- Developing a Visual Intuition for MP
- Basic Algorithms
  - MP Motif Discovery
  - MP Time Series Chains
  - MP Anomaly Discovery
  - MP Joins (self and AB)
  - MP Semantic Segmentation
- From Domain Agnostic to Domain Aware: The Annotation Vector (A simple way to use domain knowledge to adjust your results)
- The "*Matrix Profile and ten lines of code is all you need*" philosophy.
- Break

Act 2

- Background on time series mining
  - Similarity Measures
  - Normalization
- Distance Profile
  - Brute Force Approach
  - Just-in-time Normalization
  - The MASS Algorithm
  - Extensions of MASS
- Matrix Profile
  - STAMP
  - STOMP
  - GPU-STOMP
  - SCRIMP
- Open problems to solve

Distance Profile



Compute the z-normalized Euclidean distance between **Query** and each window (subsequence) in the time series. We would obtain a vector like this:

d_1	d_2	…	d_{n-m+1}

$\longrightarrow$ D

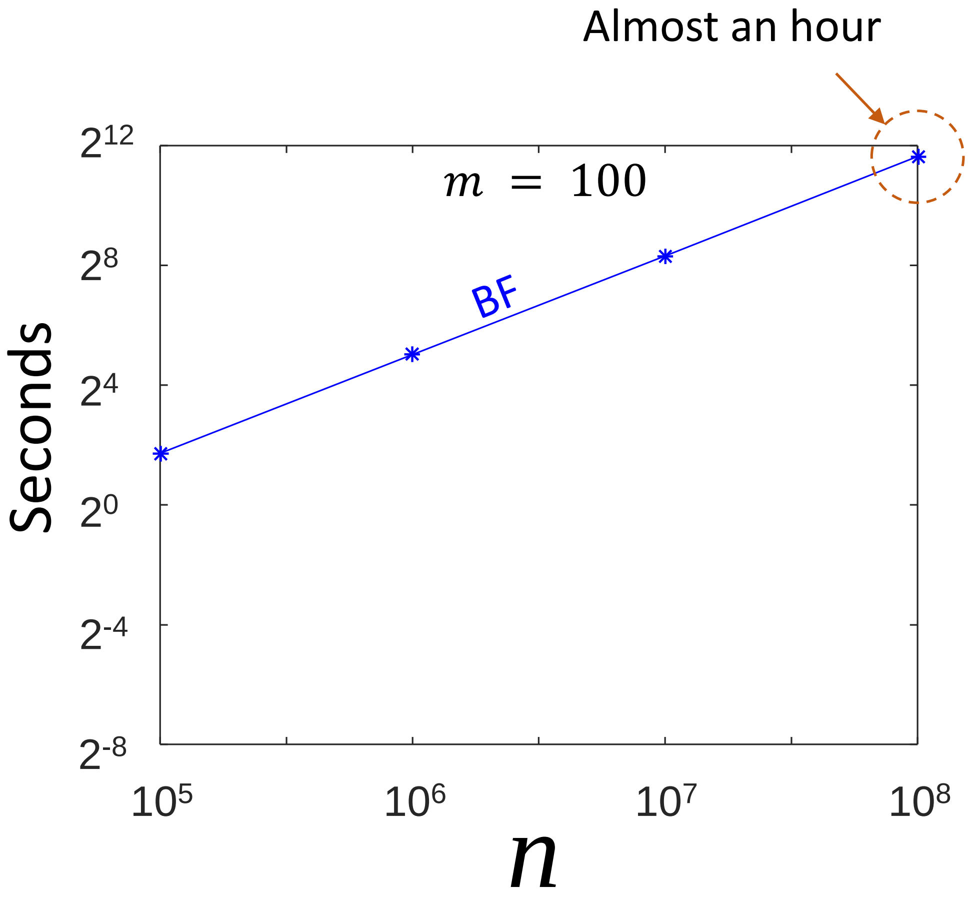
d_i is the distance between the ith subsequence and the query.

Recall, n is the length of the blue time series and m is the length of the query

The Brute Force Algorithm

- Scan the time series with a sliding window
- Z-Normalize the window
- Calculate Euclidean distance between window and the query

```
d(1:n)  = 0;
Q = zNorm(query);
for i = 1:n-m+1
    d(i) = sqrt(sum((zNorm(T(i:i+m-1))-Q).^2));
end
```



- How long does this algorithm take?
- The time complexity is $O(nm)$ in the average and worst cases. More precisely the window is scanned two times in each iteration of this algorithm. One scan is for z-normalization, and the other scan is for distance calculation.
- Note that we cannot use any early abandoning or pruning as we need all the distances.

Just-in-time Normalization (1 of 3)

- Can we skip the z-normalization scan in each iteration?
- Yes, if we have the means, standard deviations and the dot product to calculate the distances.
- z-normalized sequence has zero mean and one standard deviation.
- Let's assume y is the z-normalized query, and x is the time series (*T*), therefore, $\mu_y = 0$ and $\sigma_y = 1$

Working Formula

$$d(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) = \sqrt{2m\left(1 - \frac{\sum_{i=1}^{m} x_i y_i - m\mu_x\mu_y}{m\sigma_x\sigma_y}\right)}$$

⬇

$$d(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) = \sqrt{2\left(m - \frac{\sum_{i=1}^{m} x_i y_i}{\sigma_x}\right)}$$

Thanawin Rakthanmanon, et al. Searching and mining trillions of time series subsequences under dynamic time warping. KDD 2012: 262-270

Just-in-time Normalization (2 of 3)

- Can we skip the z-normalization scan in each iteration?
- The standard deviations of moving windows of a fixed size can be calculated in one linear scan.
  - In one pass, calculate cumulative sums of x and x^2 and store
  - Subtract two cumulative sums to obtain the sum over any window
  - Use the sums to calculate the standard deviations of all windows in linear time
- In 2016, MATLAB has introduced a function, `movstd`, that does the above.

$$d(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) = \sqrt{2(m - \frac{\sum_{i=1}^{m} x_i y_i}{\sigma_x})}$$

$$C = \sum x \qquad C^2 = \sum x^2$$

$$S_i = C_{i+m} - C_i \qquad S_i^2 = C_{i+m}^2 - C_i^2$$

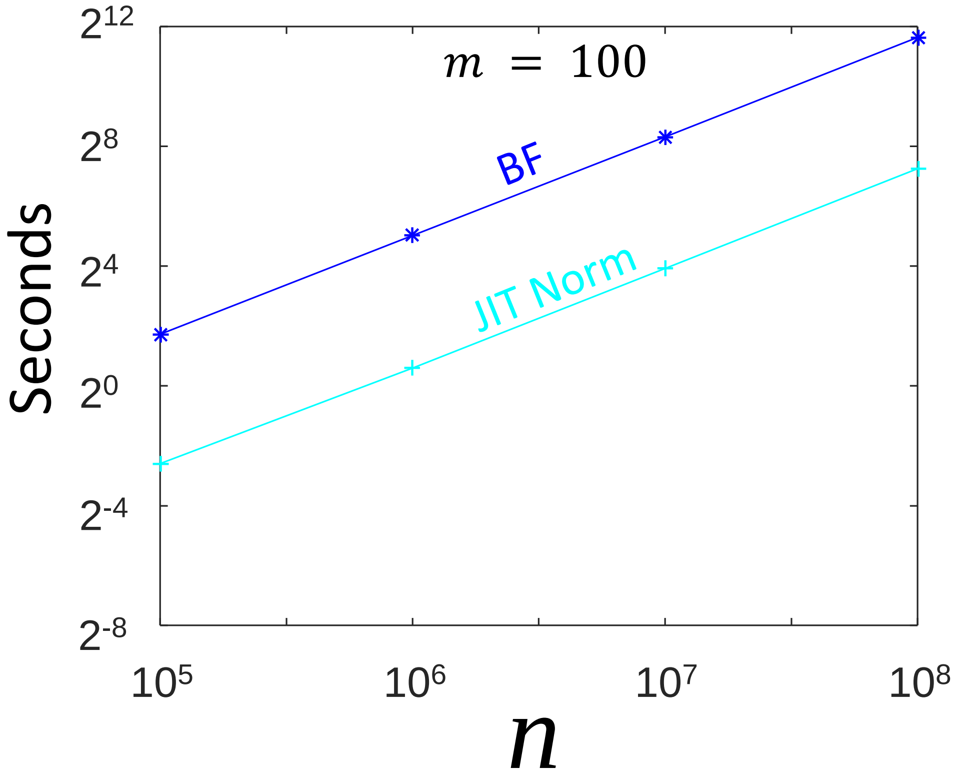
$$\sigma_i = \sqrt{\frac{S_i^2}{m} - \left(\frac{S_i}{m}\right)^2}$$

Thanawin Rakthanmanon, et al. Searching and mining trillions of time series subsequences under dynamic time warping. KDD 2012: 262-270

Just-in-time Normalization (3 of 3)

- Can we skip the z-normalization scan in each iteration?

```
d(1:n) = 0;
Q = zNorm(query);
S = movstd(T,[0 m-1]);
for i = 1:n-m+1
    d(i) = sqrt(2*(m-sum(T(i:i+m-1).*Q)/S(i)));
end
```

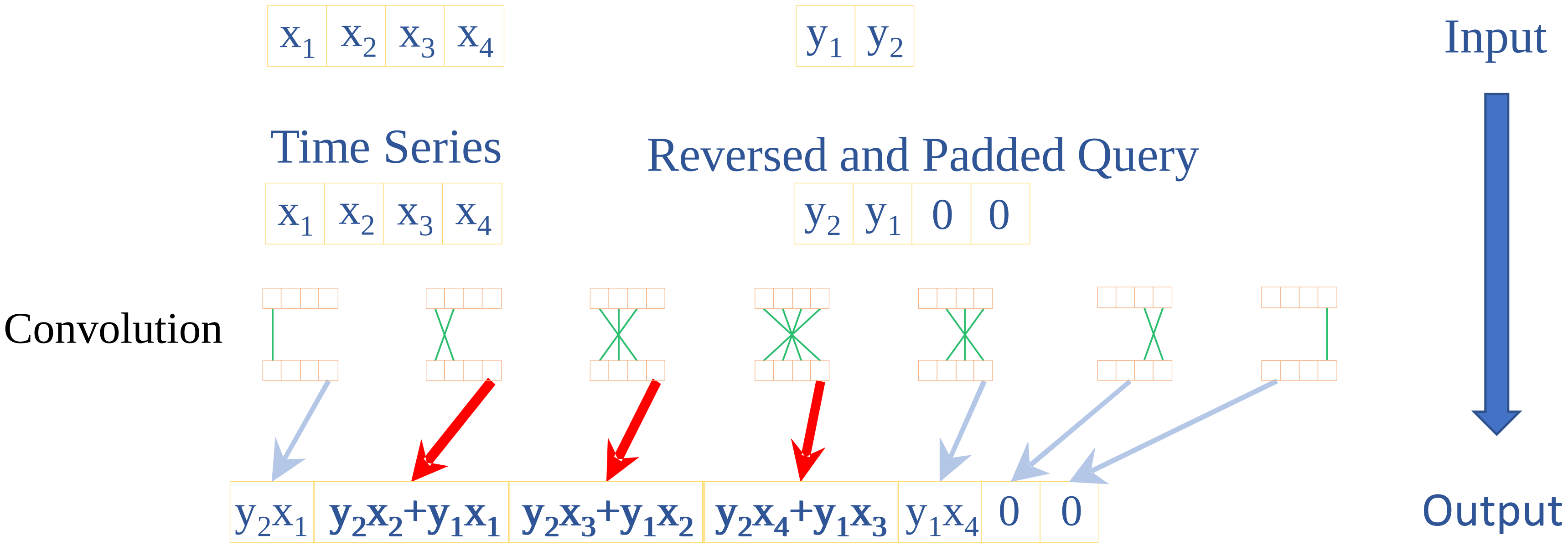


- Still the worst and average cost is $O(nm)$, however, the window is scanned only once per iteration for the dot product.
- Speedup is more than 2X, due to removal of function calls

Thanawin Rakthanmanon, et al. Searching and mining trillions of time series subsequences under dynamic time warping. KDD 2012: 262-270

Mueen’s Algorithm for Similarity Search (MASS) (1 of 9)

- Can we improve the just-in-time Normalization algorithm?
- MASS uses a convolution based method to calculate sliding dot products in $O(n \log n)$, in addition to just-in-time z-normalization technique
- **Convolution:** If x and y are vectors of polynomial coefficients, convolving them is equivalent to multiplying the two polynomials.
- We use convolution to compute all of the sliding dot products between the query and sliding windows.

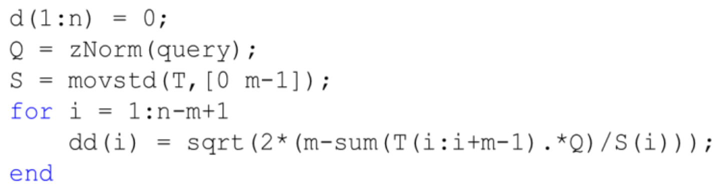


Mueen’s Algorithm for Similarity Search (MASS) (2 of 9)

MASS 1.0

```
d(1:n) = 0;
Q = zNorm(query);
Stdv = movstd(T,[0 m-1]);
Q = Q(end:-1:1);        %Reverse the query
Q(m+1:n) = 0;           %pad zeros
dots = conv(T,Q);
dist = 2*(m-(dots(m:n))./Stdv));
dist = sqrt(dist);
```

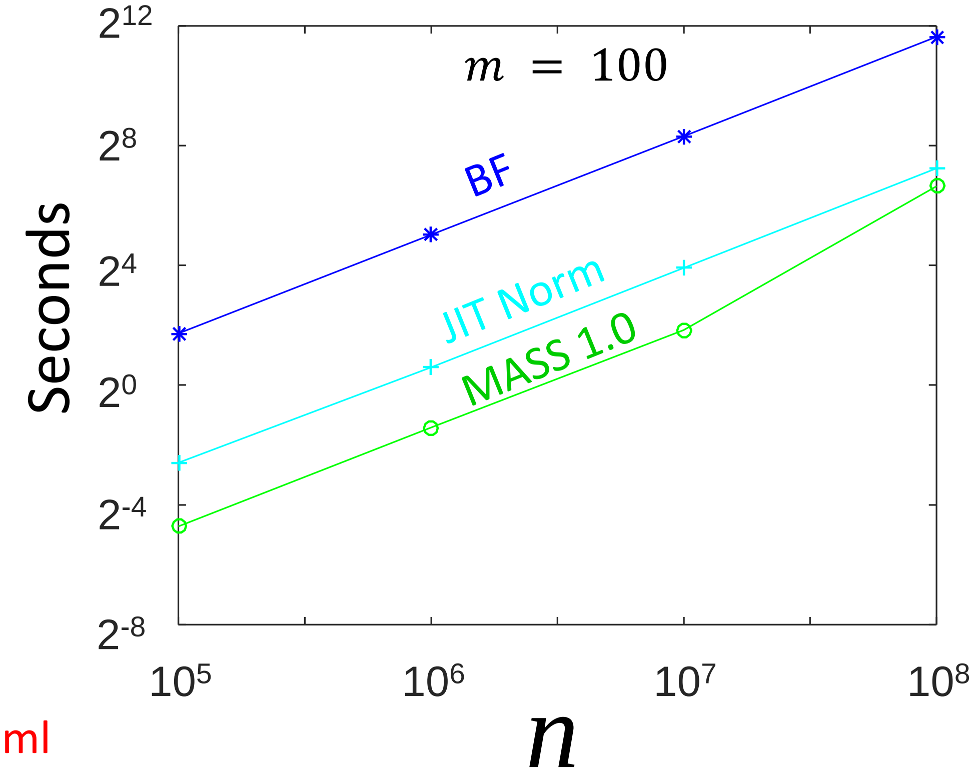
The loop has been replaced by the following three lines.



```
d(1:n) = 0;
Q = zNorm(query);
S = movstd(T,[0 m-1]);
for i = 1:n-m+1
    dd(i) = sqrt(2*(m-sum(T(i:i+m-1).*Q)/S(i)));
end
```

Vectorized working formula

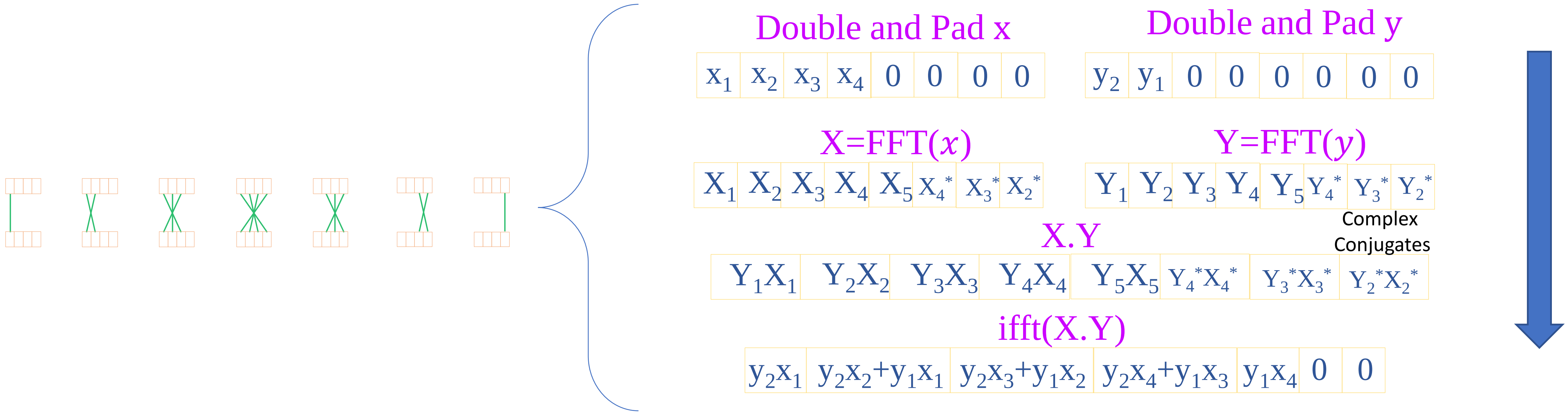
- Computational cost, $O(n \log n)$, does not depend on the query length (m), thus, free of curse of dimensionality.
- There is no loop. Only known mathematical and built-in MATLAB functions are used.



http://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html

Mueen’s Algorithm for Similarity Search (MASS) (3 of 9)

- Can we improve MASS 1.0?
- Note that convolution doubles the size of the input vectors in the output.
- MASS uses only half of the output of convolution and throws away the remaining half.
- Can we compute just the necessary half? Let’s see what happens inside convolution.
- *Convolution in time domain is multiplication in frequency domain.*
- `conv(x,y) = ifft( fft( double&pad(x)) . fft(double&pad(y))`



Double and Pad x

x_1	x_2	x_3	x_4	0	0	0	0

Double and Pad y

y_2	y_1	0	0	0	0	0	0

X=FFT(x)

X_1	X_2	X_3	X_4	X_5	X_4^*	X_3^*	X_2^*

Y=FFT(y)

Y_1	Y_2	Y_3	Y_4	Y_5	Y_4^*	Y_3^*	Y_2^*

Complex Conjugates

X.Y

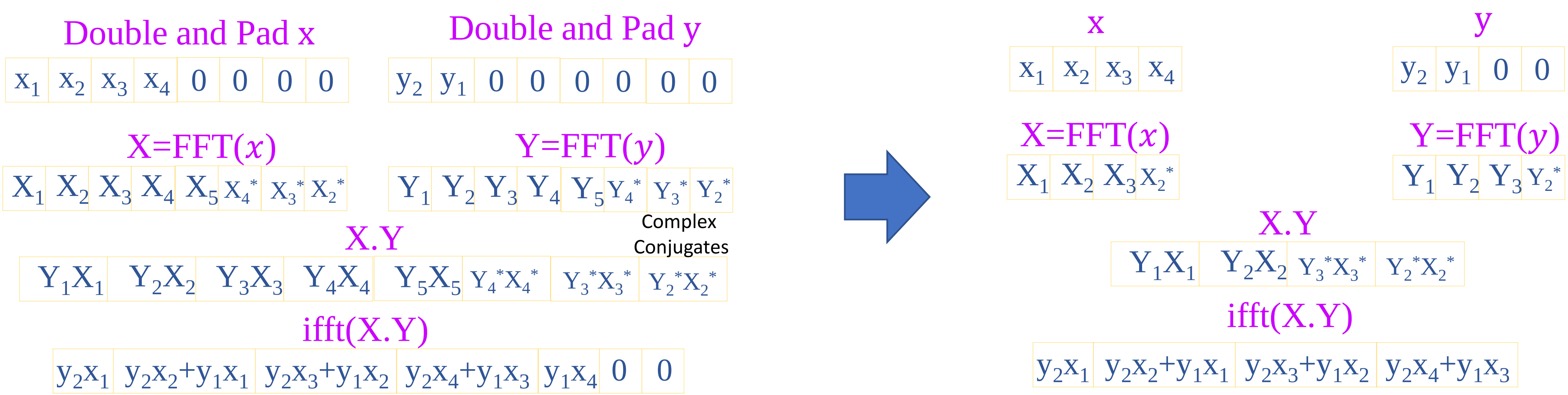
Y_1X_1	Y_2X_2	Y_3X_3	Y_4X_4	Y_5X_5	$Y_4^*X_4^*$	$Y_3^*X_3^*$	$Y_2^*X_2^*$

ifft(X.Y)

y_2x_1	$y_2x_2+y_1x_1$	$y_2x_3+y_1x_2$	$y_2x_4+y_1x_3$	y_1x_4	0	0

Mueen’s Algorithm for Similarity Search (MASS) (4 of 9)

- Can we improve MASS 1.0?
- If we do not double x and y, we obtain a half convolution
- `half conv(x,y)= ifft ( fft(x) . fft(y))`



Double and Pad x

x_1	x_2	x_3	x_4	0	0	0	0

Double and Pad y

y_2	y_1	0	0	0	0	0	0

X=FFT(*x*)

X_1	X_2	X_3	X_4	X_5	X_4^*	X_3^*	X_2^*

Y=FFT(*y*)

Y_1	Y_2	Y_3	Y_4	Y_5	Y_4^*	Y_3^*	Y_2^*

Complex Conjugates

X.Y

Y_1X_1	Y_2X_2	Y_3X_3	Y_4X_4	Y_5X_5	$Y_4^*X_4^*$	$Y_3^*X_3^*$	$Y_2^*X_2^*$

ifft(X.Y)

y_2x_1	$y_2x_2+y_1x_1$	$y_2x_3+y_1x_2$	$y_2x_4+y_1x_3$	y_1x_4	0	0

x

x_1	x_2	x_3	x_4

y

y_2	y_1	0	0

X=FFT(*x*)

X_1	X_2	X_3	X_2^*

Y=FFT(*y*)

Y_1	Y_2	Y_3	Y_2^*

X.Y

Y_1X_1	Y_2X_2	$Y_3^*X_3^*$	$Y_2^*X_2^*$

ifft(X.Y)

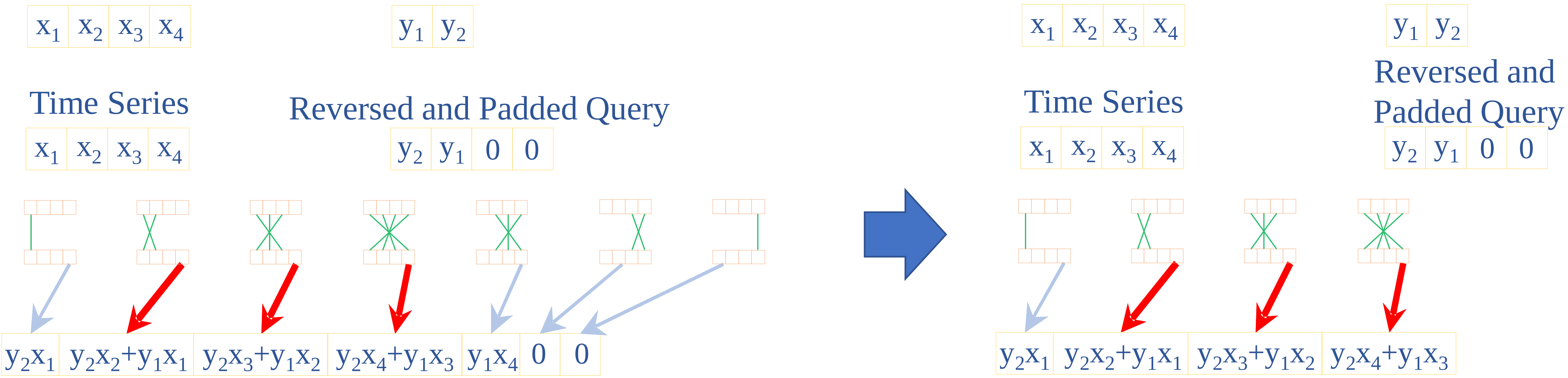
y_2x_1	$y_2x_2+y_1x_1$	$y_2x_3+y_1x_2$	$y_2x_4+y_1x_3$

```
conv(x,y) = ifft( fft( double&pad(x)) .
                       fft(double&pad(y) )
```

```
half conv(x,y)= ifft ( fft(x) . fft(y))
```

Mueen’s Algorithm for Similarity Search (MASS) (5 of 9)

- Can we improve MASS 1.0?
- Half convolution adds a constraint, $n > \frac{m}{2}$. The constraint is not limiting because the original assumption is $n \gg m$.



x_1 x_2 x_3 x_4

y_1 y_2

Time Series

Reversed and Padded Query

x_1 x_2 x_3 x_4

y_2 y_1 0 0

y_2x_1 | $y_2x_2+y_1x_1$ | $y_2x_3+y_1x_2$ | $y_2x_4+y_1x_3$ | y_1x_4 | 0 | 0

x_1 x_2 x_3 x_4

y_1 y_2

Time Series

Reversed and Padded Query

x_1 x_2 x_3 x_4

y_2 y_1 0 0

y_2x_1 | $y_2x_2+y_1x_1$ | $y_2x_3+y_1x_2$ | $y_2x_4+y_1x_3$

```
conv(x,y) = ifft( fft( double&pad(x)) .
                       fft(double&pad(y) )
```

```
half conv(x,y)= ifft ( fft(x) . fft(y))
```

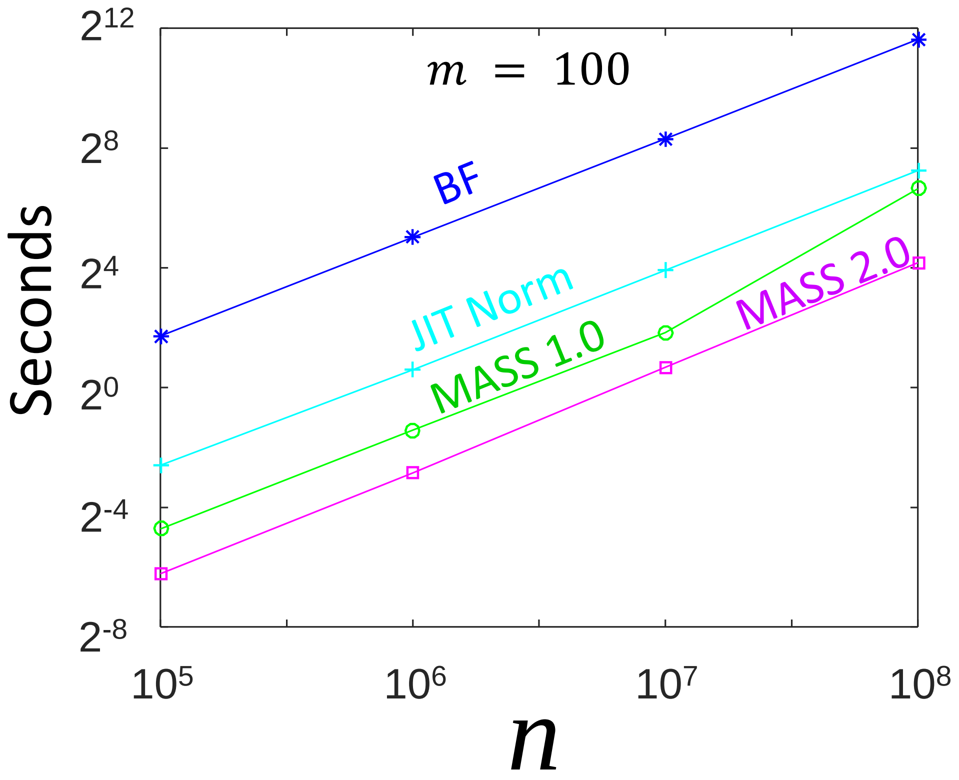
Mueen’s Algorithm for Similarity Search (MASS) (6 of 9)

MASS 2.0

```
d(1:n) = 0;
Q = zNorm(query);
Stdv = movstd(T,[0 m-1]);
Q = Q(end:-1:1);        %Reverse the query
Q(m+1:n) = 0;           %pad zeros
dots = ifft( fft(T).* fft(Q) );
dist = 2*(m-(dots(m:n))./Stdv));
dist = sqrt(dist);
```

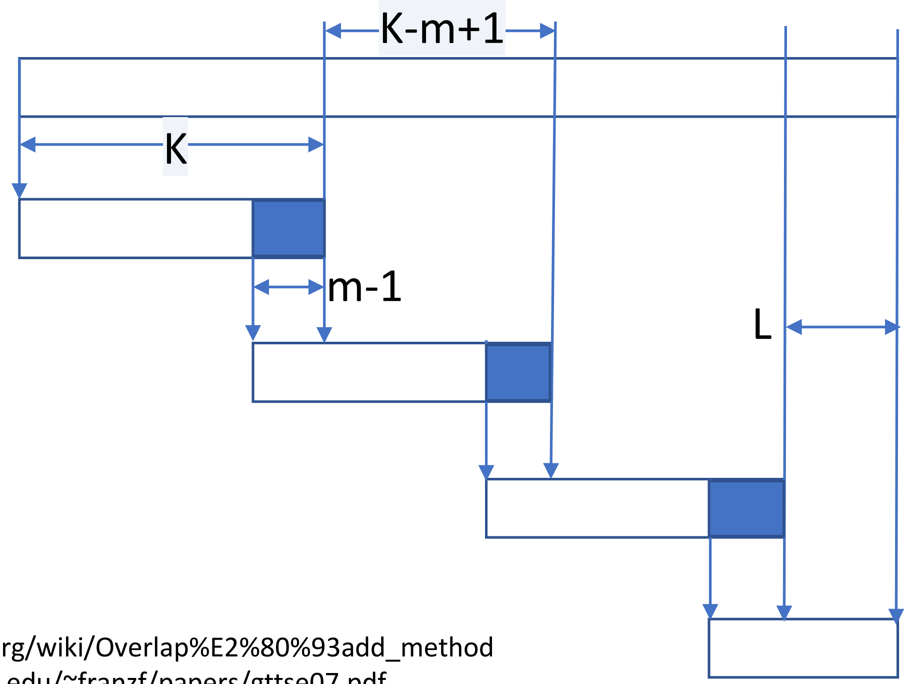
The `conv(T,Q)` has been replaced, no doubling of sizes

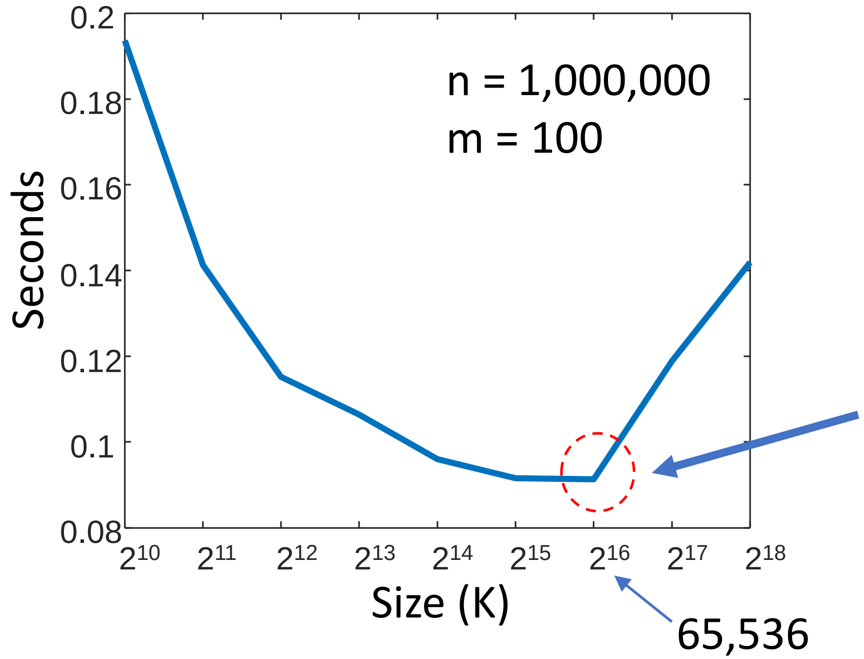
- Computational cost is still $O(n \log n)$, does not depend on the query length (m), thus, free of curse of dimensionality.
- fast Fourier transform (fft) is used as a subroutine



Mueen’s Algorithm for Similarity Search (MASS) (7 of 9)

- Can we improve MASS 2.0?
- This idea is a rediscovery of a known result, *Overlap-add*, *Overlap-save* [a]
- Instead of processing all *n* samples at once, we can (carefully) process them *piecewise,* where sizes of the pieces are powers of two.
- The pieces must overlap exactly m-1 samples to preserve continuity of the distance profile.
- The last piece can be of size that is not a power of two.

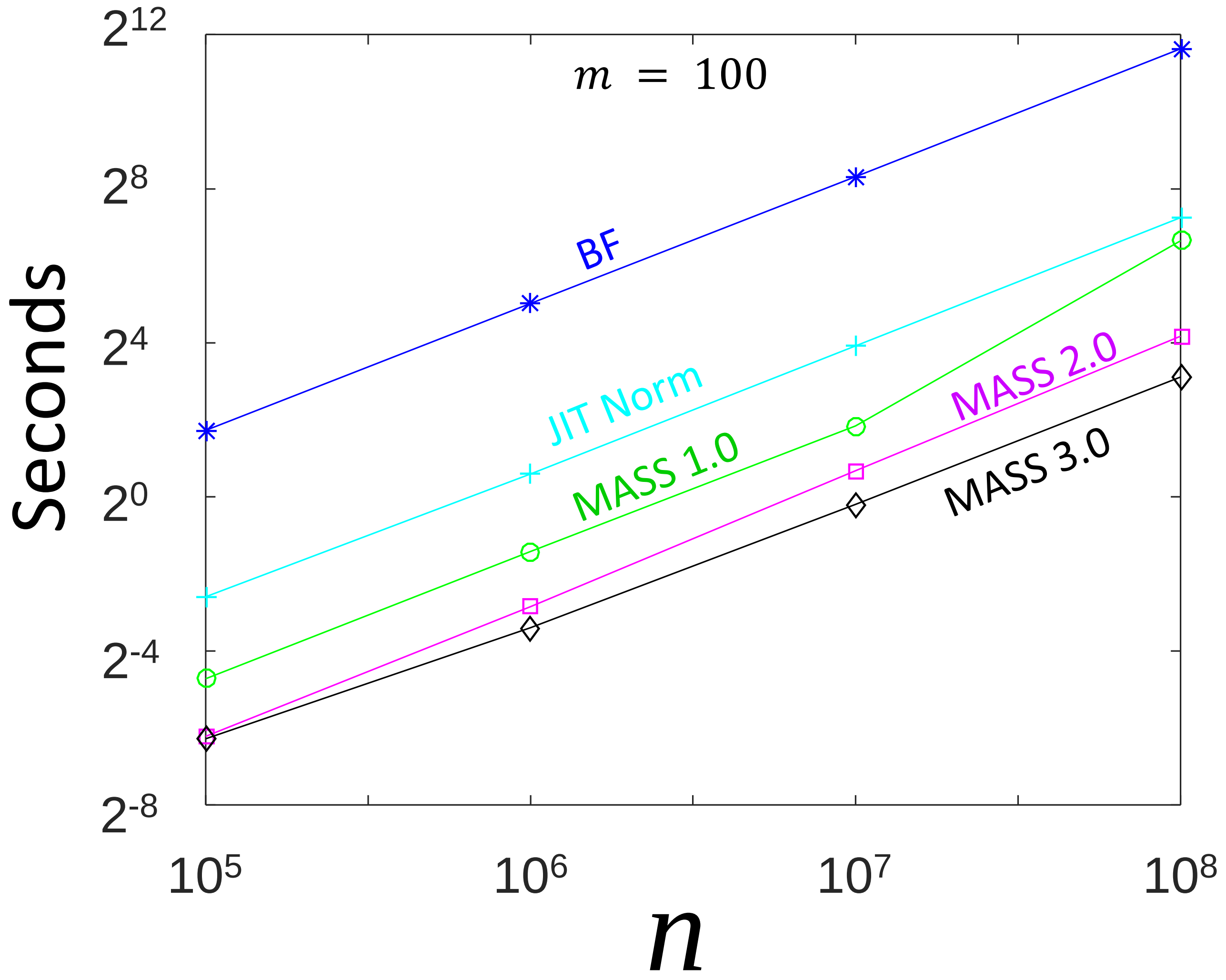




Hardware dependent minima

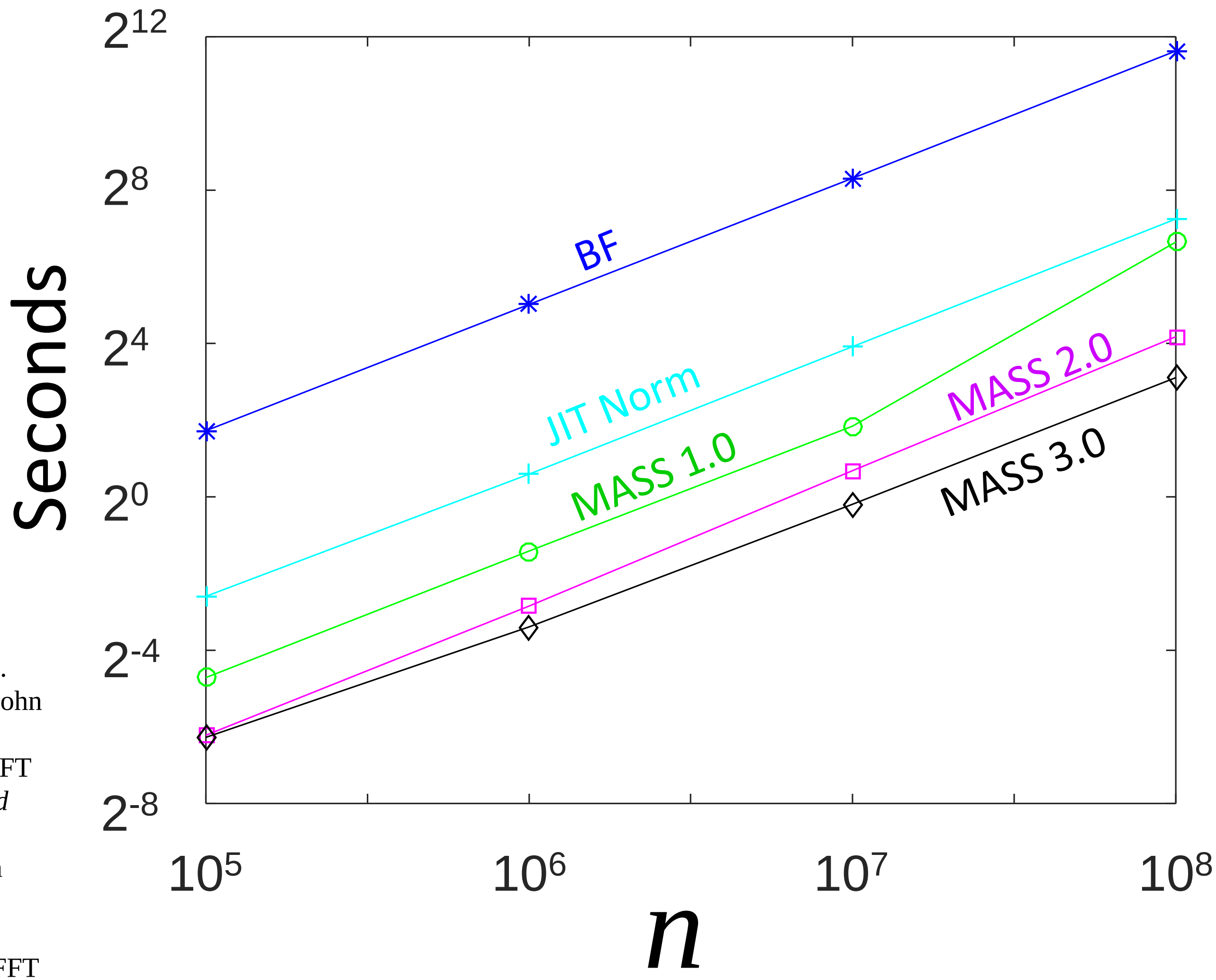
[a] https://en.wikipedia.org/wiki/Overlap%E2%80%93add_method
[b] https://users.ece.cmu.edu/~franzf/papers/gttse07.pdf

Mueen’s Algorithm for Similarity Search (MASS) (8 of 9)



Mueen’s Algorithm for Similarity Search (MASS) (9 of 9)

- Design a new application specific processor to do FFT efficiently [a][b].
- Keep the general processor and optimize DRAM to compute FFT efficiently [c].
- Keep a commodity box, add a GPU to do FFT efficiently [d].
- Keep a commodity box, add an FPGA to do FFT efficiently [e].
- Take a commodity box, and use sparse FFT [f].



[a] The Fast Fourier Transform in Hardware: A Tutorial Based on an FPGA Implementation, George Slade.
[b] Appendix C: Efficient Hardware Implementations of FFT Engines, Nasserbakht, Mitra (Ed. Bingham, John A. C.) ADSL, VDSL, and Multicarrier Modulation, John Wiley & Sons, Inc. 2001
[c] B. Akın, F. Franchetti and J. C. Hoe, "Understanding the design space of DRAM-optimized hardware FFT accelerators," *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*, Zurich, 2014, pp. 248-255.
[d] Naga K. Govindaraju, Brandon Lloyd, Yuri Dotsenko, Burton Smith, and John Manferdelli. 2008. High performance discrete Fourier transforms on graphics processors. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (SC '08).
[e] Ren Chen, Shreyas G. Singapura, and Viktor K. Prasanna. 2017. Optimal dynamic data layouts for 2D FFT on 3D memory integrated FPGA. *J. Supercomput.* 73, 2 (February 2017), 652-663.
[f] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. 2012. Simple and practical algorithm for sparse Fourier transform. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete algorithms* (SODA '12). 1183-1194.

Numerical Errors in MASS

- Three possible sources of numerical errors in MASS
- Convolution
    - Numerical error in Convolution can appear if n is very large, MASS 3.0 reduces such error by dividing the computation in batches
- Division by the standard deviation
    - If a subsequence is constant, the standard deviation is zero, causing divide by zero errors. MASS reduces such error by checking the standard deviations ahead of the division.
- Square root of the distances
    - Theoretically it is not possible to have negative squared distance. However, for exact matches, the squared distance can be a very small negative number (-1.09e-15), resulting imaginary numbers in the output.

```
d(1:n) = 0;
Q = zNorm(query);
Stdv = movstd(T,[0 m-1]);
Q = Q(end:-1:1);
Q(m+1:n) = 0;
dots = ifft( fft(T).* fft(Q) );
dist = 2*(m-(dots(m:n))./Stdv));
dist = sqrt(dist);
```

Act 1

Outline

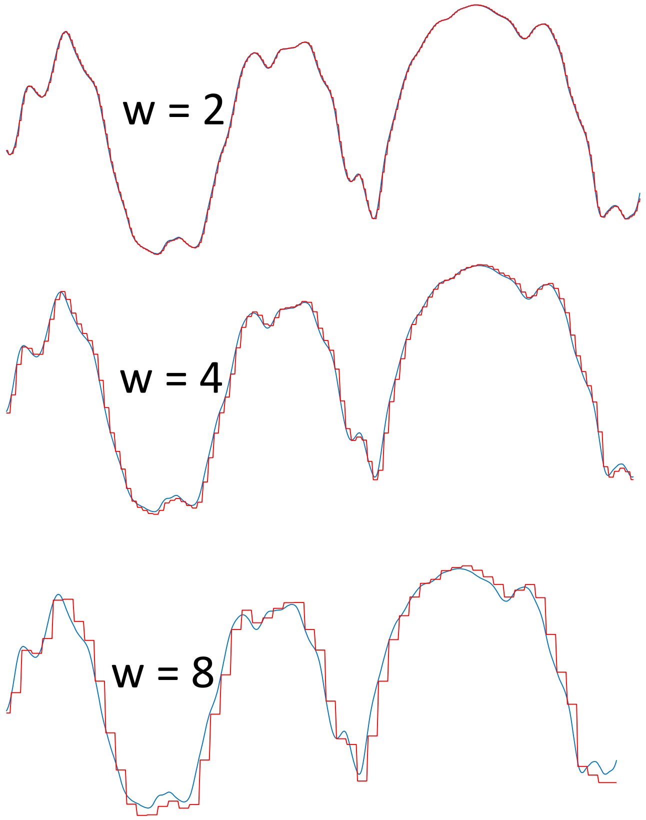
Act 2

- Our Fundamental Assumption
- What is the (MP) Matrix Profile?
- Properties of the MP
- Developing a Visual Intuition for MP
- Basic Algorithms
  - MP Motif Discovery
  - MP Time Series Chains
  - MP Anomaly Discovery
  - MP Joins (self and AB)
  - MP Semantic Segmentation
- From Domain Agnostic to Domain Aware: The Annotation Vector (A simple way to use domain knowledge to adjust your results)
- The "*Matrix Profile and ten lines of code is all you need*" philosophy.
- Break

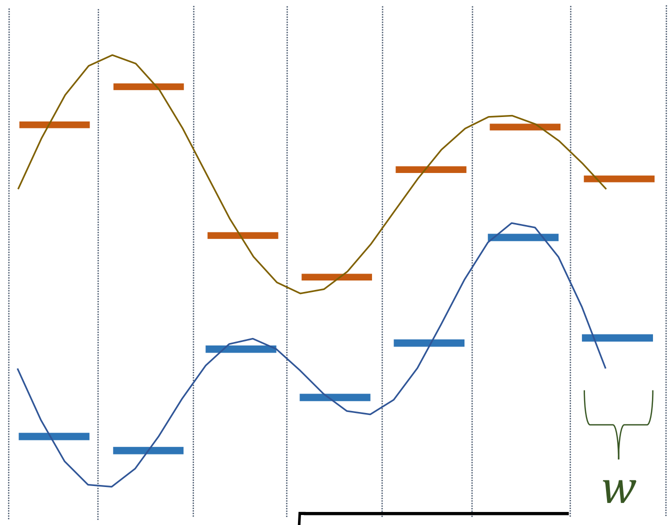
- Background on time series mining
  - Similarity Measures
  - Normalization
- Distance Profile
  - Brute Force Approach
  - Just-in-time Normalization
  - The MASS Algorithm
  - Extensions of MASS ← Rest Zone
- Matrix Profile
  - STAMP
  - STOMP
  - GPU-STOMP
  - SCRIMP
- Open problems to solve

MASS Extensions: Approximate Distance Profile

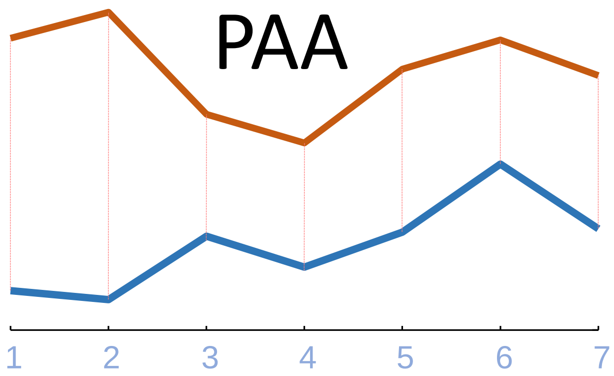
- Can we trade speed with accuracy in MASS?
- We use piecewise aggregate approximation (PAA) to reduce dimensionality of both T and Q
- `sqrt(w)* MASS(PAA(T,w),PAA(Q,w),m/w)`



Approximate distance profile in Red



$$d(x,y) = \sqrt{\sum_{i=1}^{m}(x_i - y_i)^2}$$



$$LB(x,y) = \sqrt{w\sum_{i=1}^{m/w}(x_i - y_i)^2}$$

Selina Chu, Eamonn J. Keogh, David M. Hart, Michael J. Pazzani: Iterative Deepening Dynamic Time Warping for Time Series. SDM 2002: 195-212

MASS Extensions: Weighted Query

Given two time series

$$\boldsymbol{x} = x_1 \ldots x_n$$

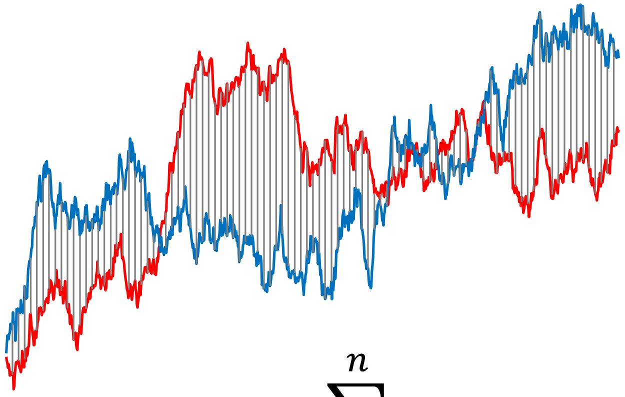
and

$$\boldsymbol{y} = y_1 \ldots y_n$$

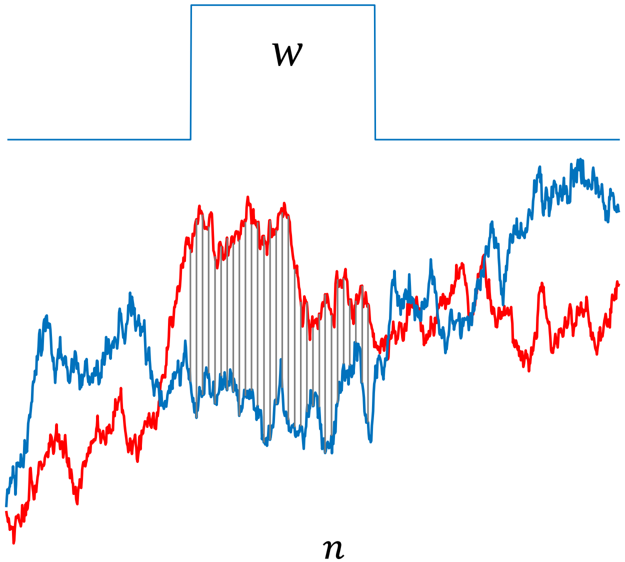
their z-Normalized weighted Euclidean distance is defined as:

$$\widehat{x_i} = \frac{x_i - \mu_x}{\sigma_x} \qquad \widehat{y_i} = \frac{y_i - \mu_y}{\sigma_y} \qquad d^2(x,y) = \sum_{i=1}^{n} w_i (\widehat{x_i} - \widehat{y_i})^2$$

- $|w| = m$
- $\forall\, w_i \geq 0$
- $\sum w_i = 1$



$$d(x,y) = \sum_{i=1}^{n} (\widehat{x_i} - \widehat{y_i})^2$$



$$d^2(x,y) = \sum_{i=1}^{n} w_i (\widehat{x_i} - \widehat{y_i})^2$$

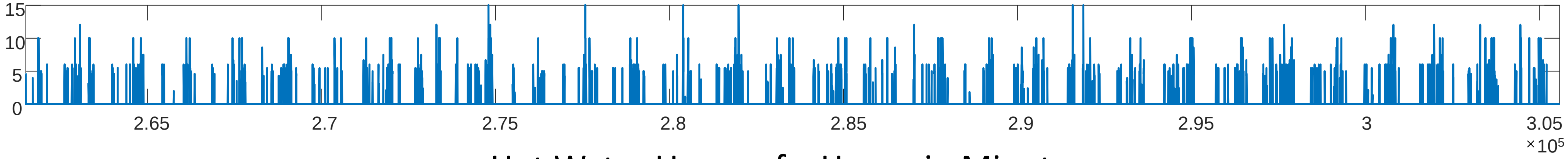
MASS Extensions: Weighted Query

- Working formula that we have been using so far, will not work anymore.
- The new working formula is below.
- We need three sliding dot products instead of *one* in regular MASS

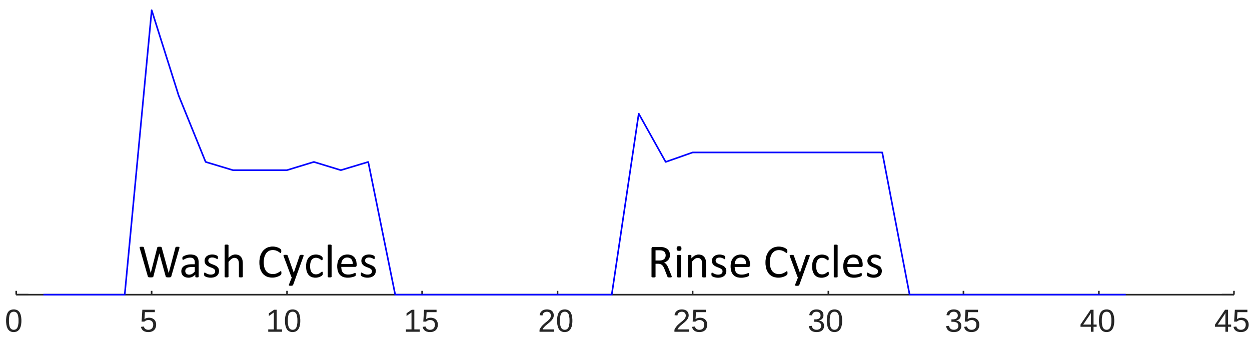
$$d^2(x,y) = \sum_{i=1}^{n} w_i(\widehat{x_i} - \widehat{y_i})^2 = \sum_{i=1}^{n} w_i(\frac{x_i - \mu_x}{\sigma_x} - \widehat{y_i})^2$$

$$= \sum_{i=1}^{n} \underbrace{w_i x_i^2 - 2\mu_x w_i x_i - 2w_i x_i\,\widehat{y_i}}_{\text{Terms with } x \text{ need sliding dot products}} + \underbrace{\mu_x^2 w_i - 2\mu_x w_i \widehat{y_i} + w_i \widehat{y_i}^{\,2}}_{\text{Terms without } x \text{ are precomputed}}$$

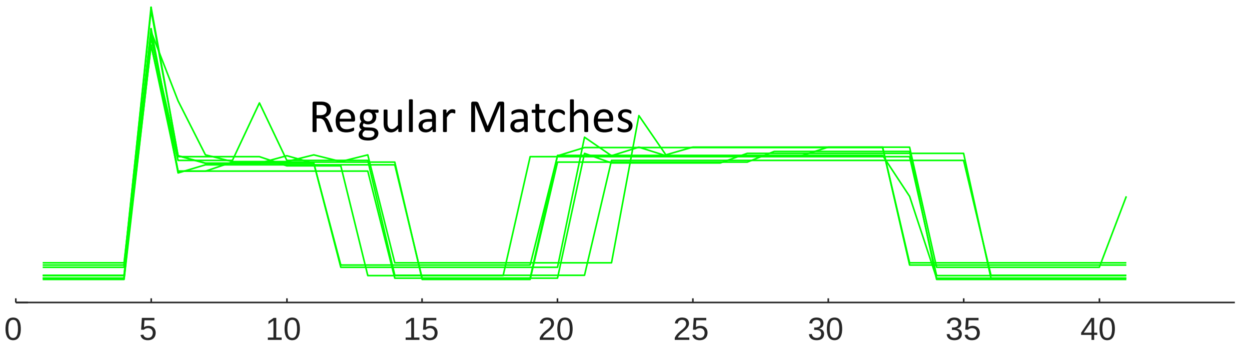
MASS Extensions: Weighted Query Example

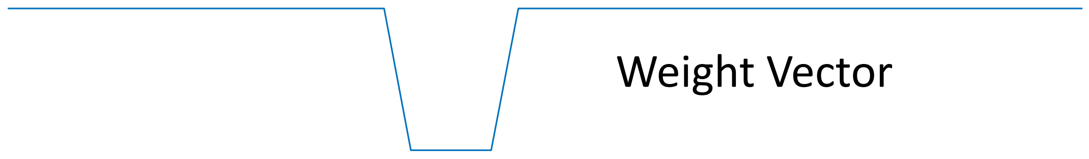


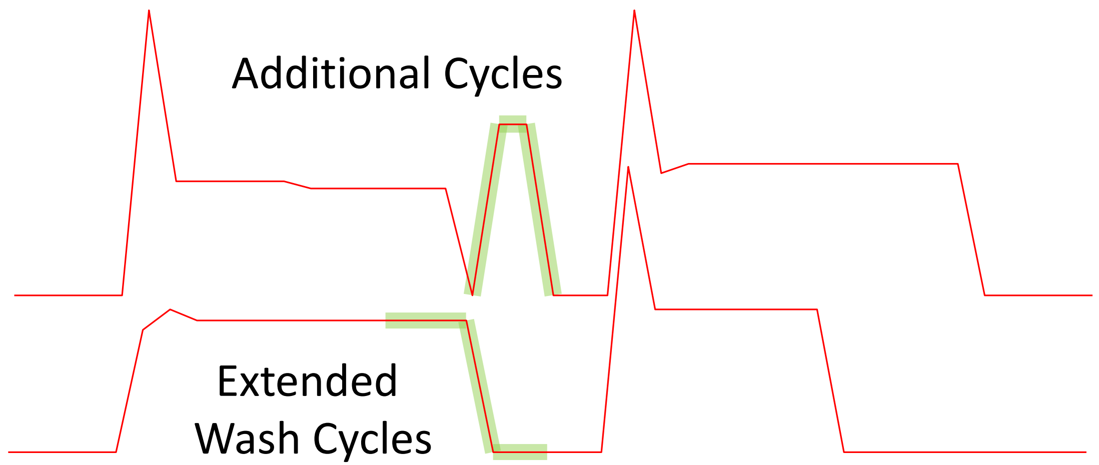
Hot Water Usage of a House in Minutes



Washer







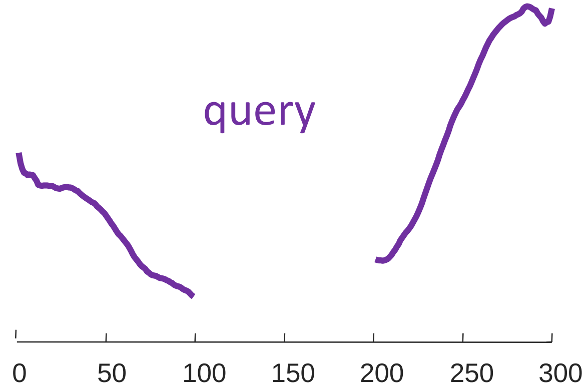
MASS Extensions: Query with Gap(s)

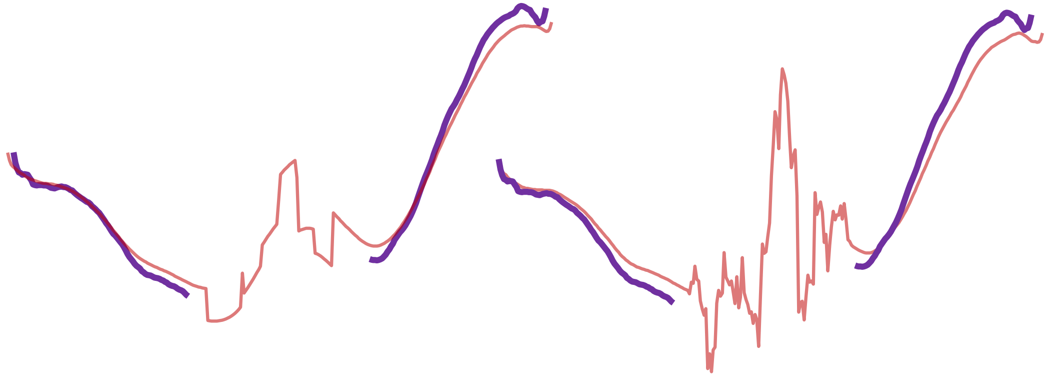
We may wish to consider a more general type of query. Queries with gaps, as in the purple example.

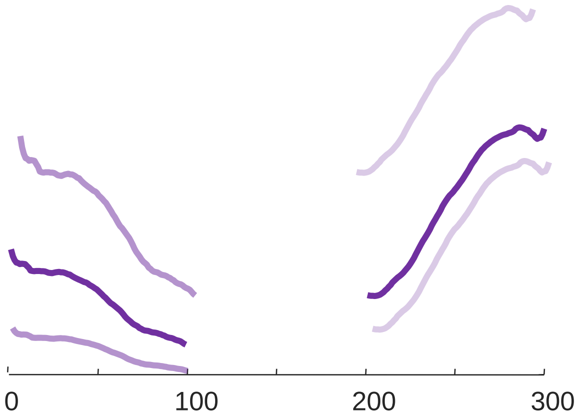
I want the first third to slope down, and the last third to slope up with the plateau at the end. However, I don't care what goes in the middle.

Either of the two red patterns would be acceptable matches.

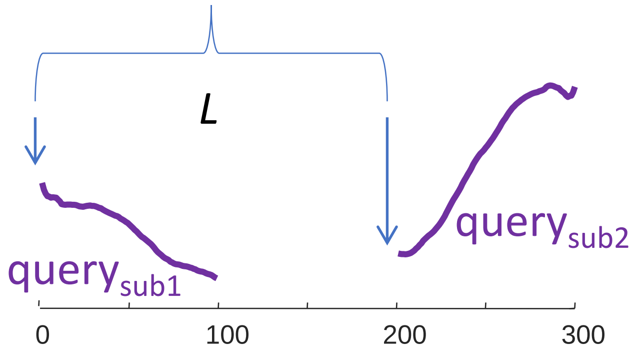
Note that the relative offset and scaling of the two parts do *not* matter, and we should be invariant to them. In other words, the exact slope does not matter in the above query statement.







MASS Extensions: Query with Gap(s)



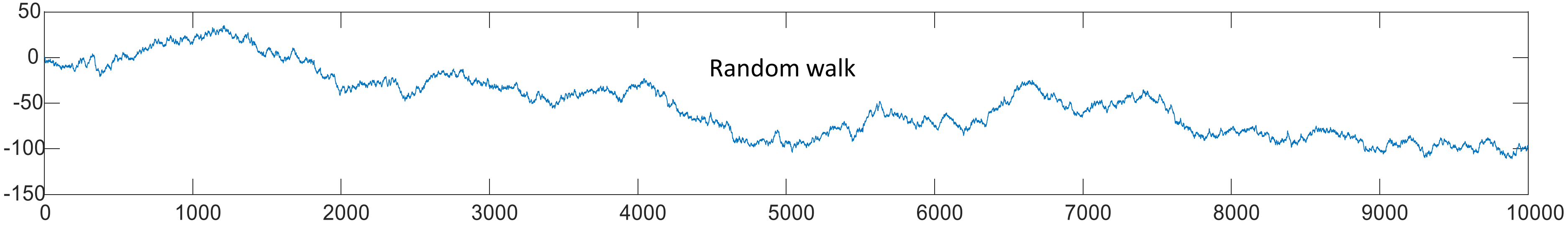
1. Measure the distance between beginning of the two patterns, call it *L*
2. Compute…

   `Dist_profile1 = MASS(T,` query$_{sub1}$`);`

   `Dist_profile2 = MSAS(T,` query$_{sub2}$`);`
3. Pad the *end* of `Dist_profile1` and the *beginning* of `Dist_profile2` with a vector of *L* infinities to add a lag of *L*

   For example, if *L* = 5, then use {inf, inf, inf, inf, inf}
4. Create `Dist_profile3 = Dist_profile1 + Dist_profile2`
5. We can now use `Dist_profile3` to find the nearest neighbor(s) as before.

MASS Extensions: Query with Gap(s)



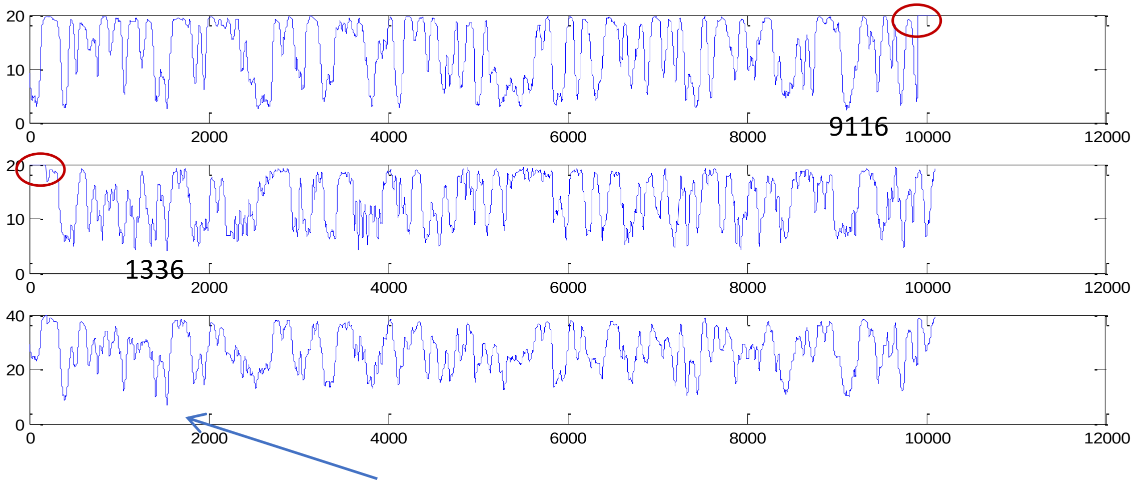
Dist_profile1 = MASS(tag, $\text{query}_{\text{sub1}}$);

Slide by L slots

Dist_profile2 = MASS(tag, $\text{query}_{\text{sub2}}$);

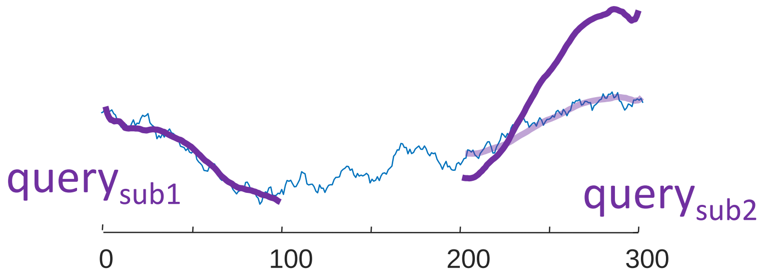
Dist_profile3 = Dist_profile1 + Dist_profile2

Padded by *L* infs or NaNs



Min value is at 1536

Note that for the best match, each part of the query scaled differently.
For the image to the right, I fixed the left side and scaled the right side (to give context), but both sides needed rescaling somewhat



Outline

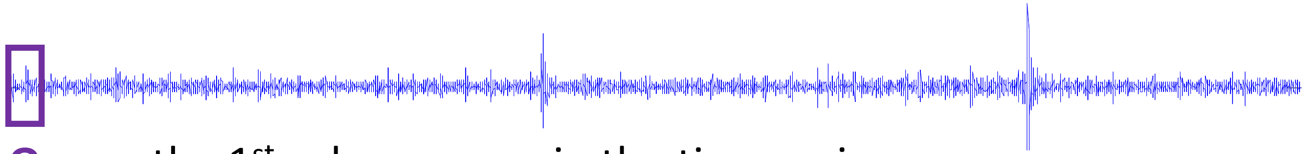
Act 1

- Our Fundamental Assumption
- What is the (MP) Matrix Profile?
- Properties of the MP
- Developing a Visual Intuition for MP
- Basic Algorithms
  - MP Motif Discovery
  - MP Time Series Chains
  - MP Anomaly Discovery
  - MP Joins (self and AB)
  - MP Semantic Segmentation
- From Domain Agnostic to Domain Aware: The Annotation Vector (A simple way to use domain knowledge to adjust your results)
- The "*Matrix Profile and ten lines of code is all you need*" philosophy.
- Break

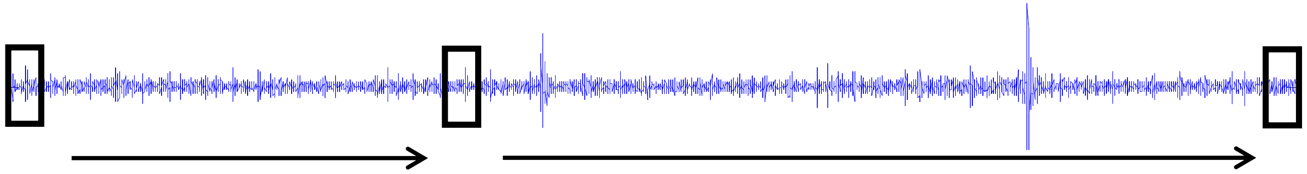
Act 2

- Background on time series mining
  - Similarity Measures
  - Normalization
- Distance Profile
  - Brute Force Approach
  - Just-in-time Normalization
  - The MASS Algorithm
  - Extensions of MASS
- Matrix Profile
  - STAMP
  - STOMP
  - GPU-STOMP
  - SCRIMP
- Open problems to solve

Matrix Profile from Distance Profiles



Query, the 1st subsequence in the time series



Obtain the z-normalized Euclidean distance between **Query** and each window (subsequence) in the time series. We would obtain a vector like this:

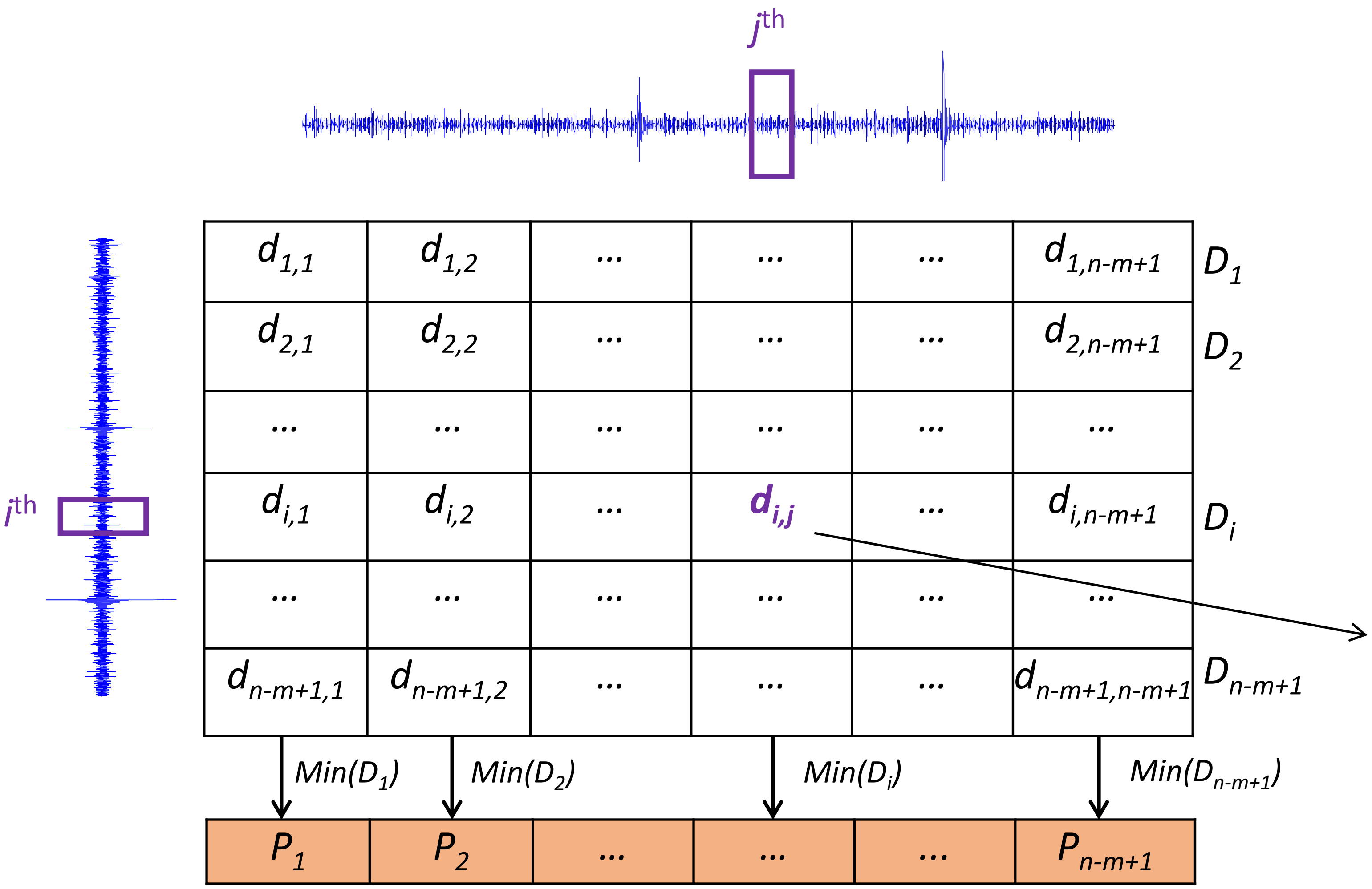
$d_{1,1}$(= 0)	$d_{2,1}$	…	$d_{n-m+1,1}$

$\longrightarrow$ D_1

$d_{i,j}$ is the distance between the ith subsequence and the jth subsequence.

We can obtain D_2, D_3, … D_{n-m+1} similarly.

Matrix Profile from Distance Profiles



$d_{1,1}$	$d_{1,2}$	…	…	…	$d_{1,n-m+1}$	D_1
$d_{2,1}$	$d_{2,2}$	…	…	…	$d_{2,n-m+1}$	D_2
…	…	…	…	…	…	
$d_{i,1}$	$d_{i,2}$	…	$\boldsymbol{d_{i,j}}$	…	$d_{i,n-m+1}$	D_i
…	…	…	…	…	…	
$d_{n-m+1,1}$	$d_{n-m+1,2}$	…	…	…	$d_{n-m+1,n-m+1}$	D_{n-m+1}

$Min(D_1)$ $Min(D_2)$ $Min(D_i)$ $Min(D_{n-m+1})$

P_1	P_2	…	…	…	P_{n-m+1}

The distance matrix is symmetric

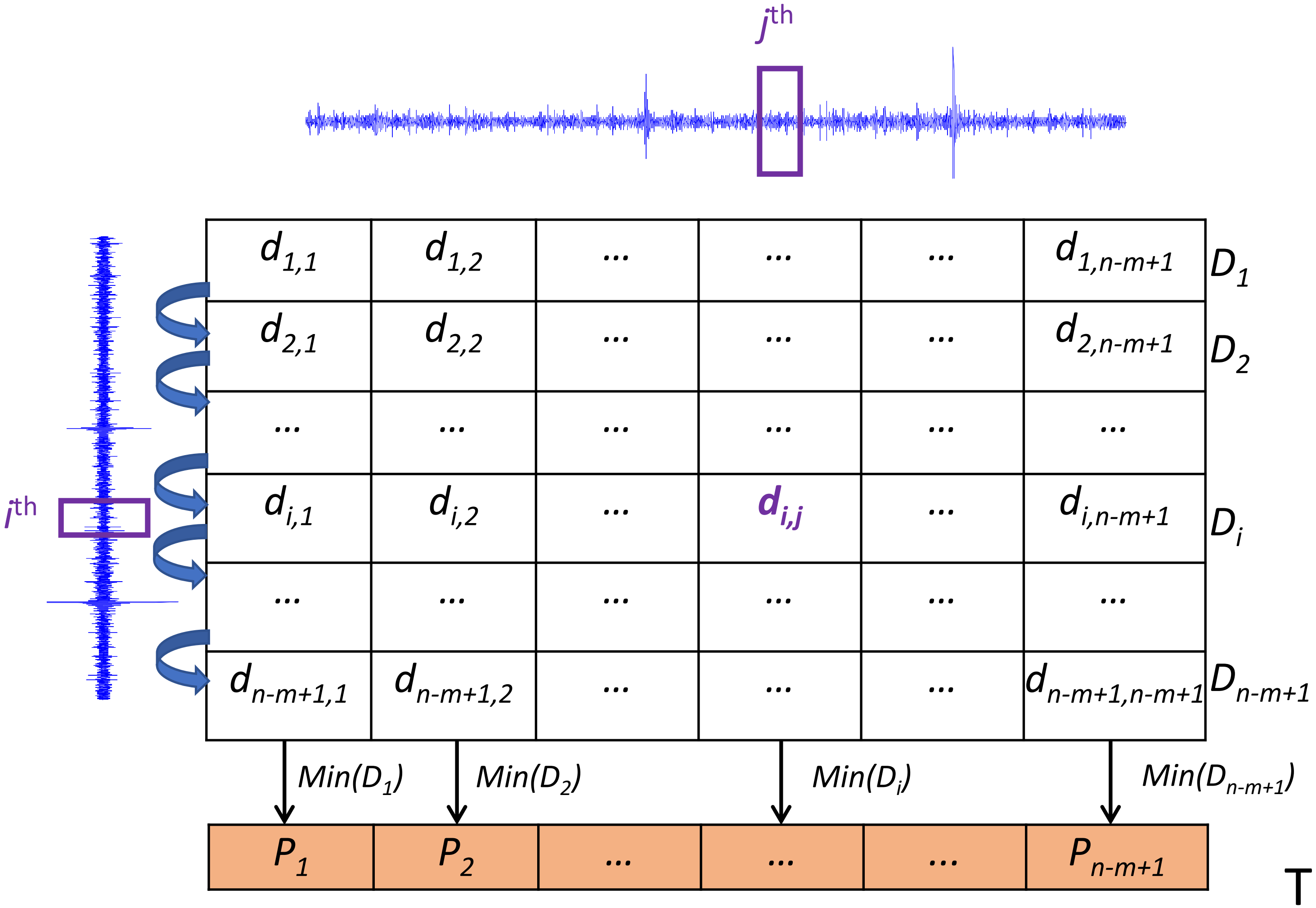
The diagonal is zero

Cells close to the diagonal are very small

$d_{i,j}$ is the distance between the i^{th} window and the j^{th} window of the time series

Matrix Profile: a vector of distance between each subsequence and its nearest neighbor

STMP: Scalable Time Series Matrix Profile Algorithm



$d_{1,1}$	$d_{1,2}$	…	…	…	$d_{1,n-m+1}$	D_1
$d_{2,1}$	$d_{2,2}$	…	…	…	$d_{2,n-m+1}$	D_2
…	…	…	…	…	…	
$d_{i,1}$	$d_{i,2}$	…	$d_{i,j}$	…	$d_{i,n-m+1}$	D_i
…	…	…	…	…	…	
$d_{n-m+1,1}$	$d_{n-m+1,2}$	…	…	…	$d_{n-m+1,n-m+1}$	D_{n-m+1}

P_1	P_2	…	…	…	P_{n-m+1}

```
MP(1:n-m+1) = inf;
for i = 1:n-m+1
        d = MASS(T,T(i:i+m-1));
        MP = min([MP ; d]);
end
```

Matrix Profile: a vector of distance between each subsequence and its nearest neighbor

Time complexity of STMP is $O(n^2 \log n\ )$

Space complexity of STMP is $O(n)$

Matrix Profile Index

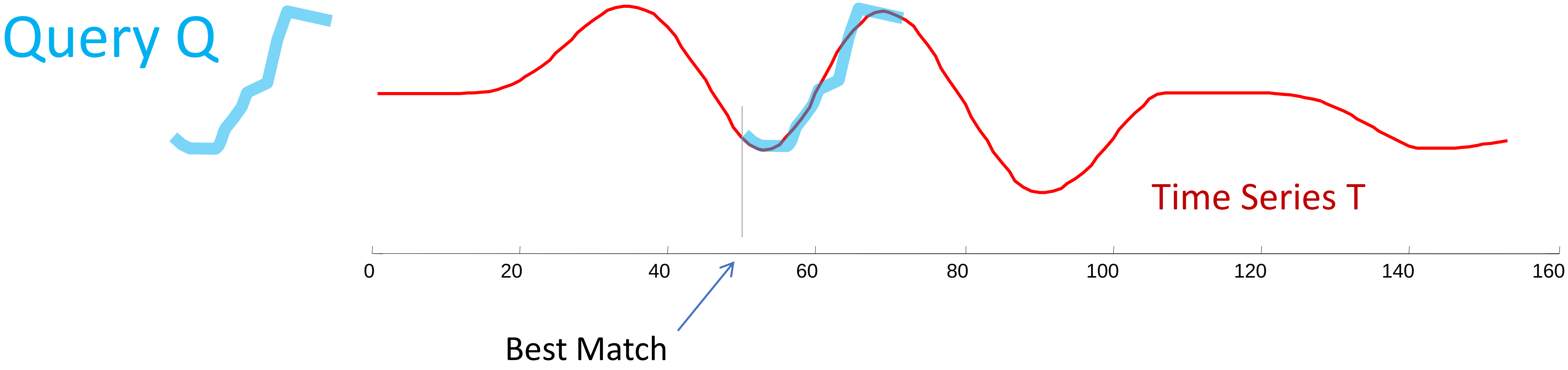
- Matrix Profile Index (MPI) can be maintained as we compute the profile
- Vectorized `min` functions can be used to efficiently maintain MPI
- Overhead is negligible

```
MP(1:n-m+1) = inf;
MPI(1:n-m+1) = -1;
for i = 1:n-m+1 in a random order
    d = MASS(T,T(i:i+m-1));
    [MP, ind] = min([MP ; d]);
    MPI(ind==2) = i;
end
```

Assumes `MP` and `d` as column vectors

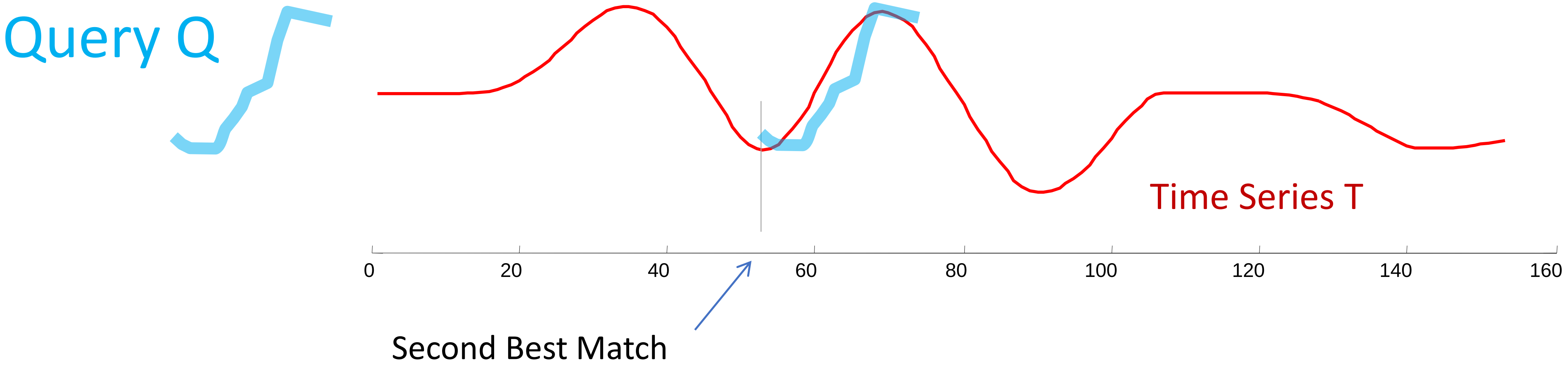
Trivial Match I

- We need to understand the idea of a "trivial match". It shows up for definitions of discords, motifs, similarity search etc.
- Suppose we search for the query, of length 20, in the time series…
- …we find its best match is at location 50…



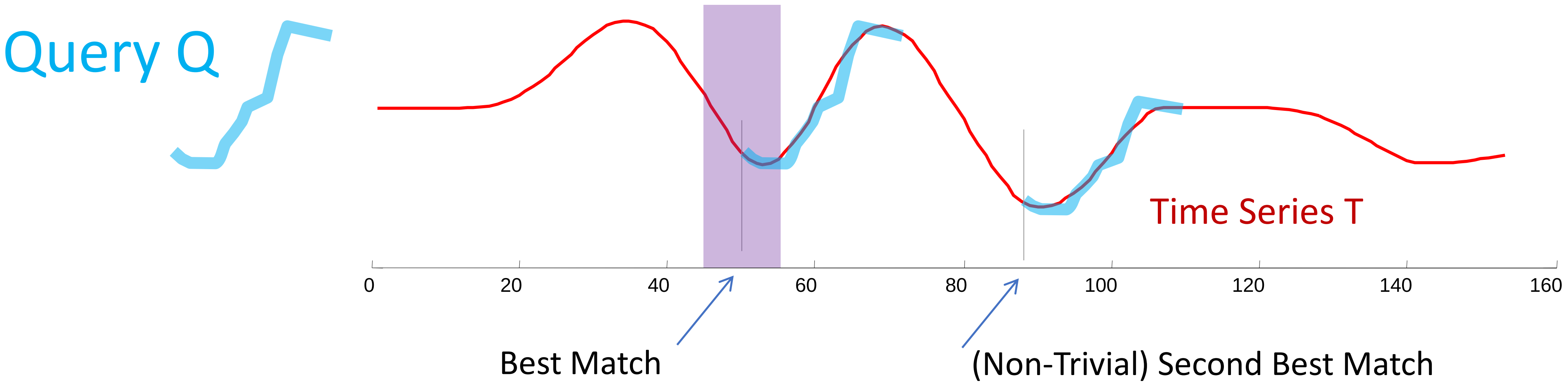
Trivial Match II

- Where is the second best match? It is probably going to be at 49 or 51, but that is *trivial*, it is just a minor “variant” of the original match.
- (try togging backwards and forwards between this and the last slide)



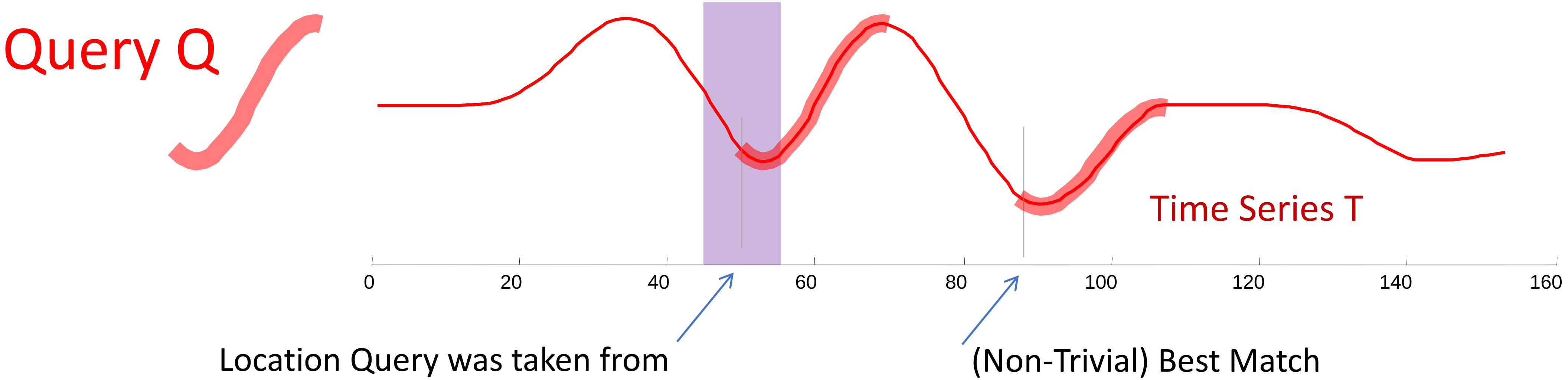
Trivial Match III

- To avoid trivial matches, we find the first match, then we set an *exclusion zone* around the best match, then find the *second* best match, etc.
- The size of the *exclusion zone* is not critical, ½ *m* works well.



Trivial Match IIII   Special Case

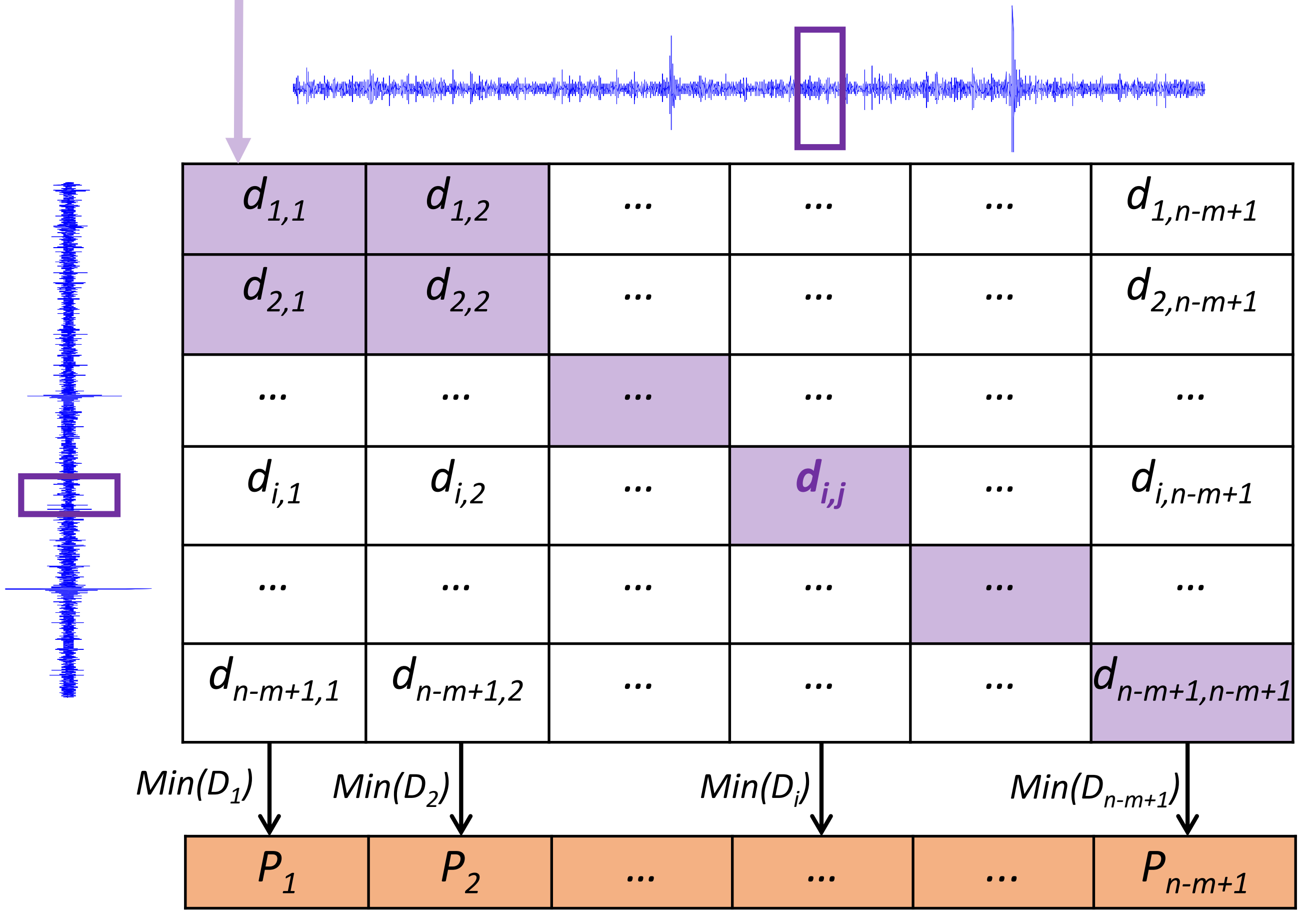
- When computing the MP, we will be extracting the subsequences from the time series *itself*.
- Clearly such queries will just find "themselves" as their own nearest neighbor!
- The distance from the query to any part of the subsequence that overlaps the *exclusion zone* is kept undefined (NaN).



STMP: Scalable Time Series Matrix Profile Algorithm

```
MP(1:n-m+1) = inf;
MPI(1:n-m+1) = -1;
for i = 1:n-m+1
      d = MASS(T,T(i:i+m-1));
      d( max(i-m/4,1) : min(i+m/4-1,n-m+1))= NaN;
      [MP, ind] = min([MP ; d]);
      MPI(ind==2) = i;
end
```

Size of Exclusion Zone depends on the smoothness of the time series. A default value of m/2 works well. Data adaptive heuristic could be to set the length of the exclusion zone equal to *m* or the period (1/f) of the highest energy frequency, whichever is smaller.



$d_{1,1}$	$d_{1,2}$	…	…	…	$d_{1,n-m+1}$
$d_{2,1}$	$d_{2,2}$	…	…	…	$d_{2,n-m+1}$
…	…	…	…	…	…
$d_{i,1}$	$d_{i,2}$	…	$d_{i,j}$	…	$d_{i,n-m+1}$
…	…	…	…	…	…
$d_{n-m+1,1}$	$d_{n-m+1,2}$	…	…	…	$d_{n-m+1,n-m+1}$

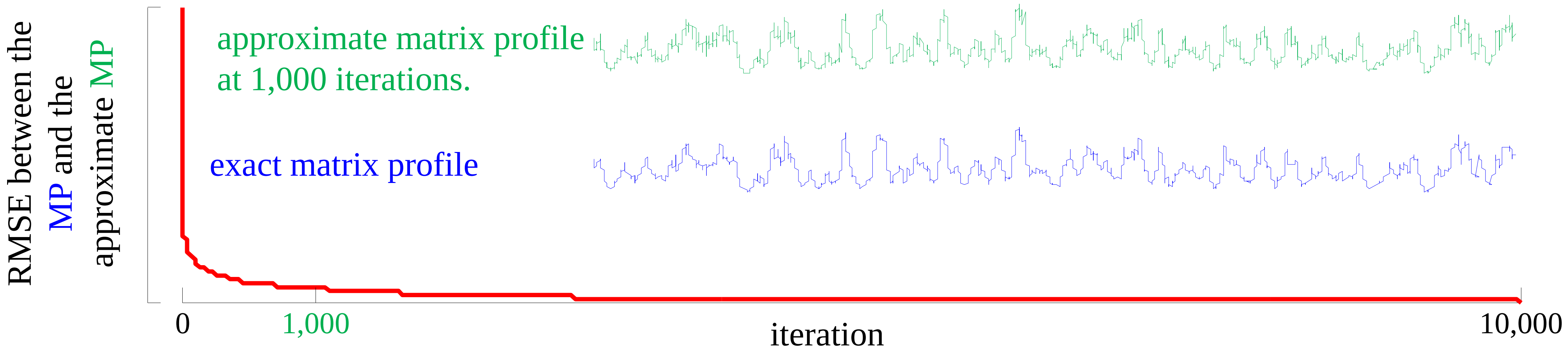
P_1	P_2	…	…	…	P_{n-m+1}

STAMP: Scalable Time Series Anytime Matrix Profile

- Each distance profile is independent of other distance profiles, the order in which we compute them can be random
- A random ordering averages the performance over properties of the time series, such as smoothness, frequency, stationarity, etc.
- The random ordering provides diminishing return, which allows *interrupt-resume* operations *anytime*.

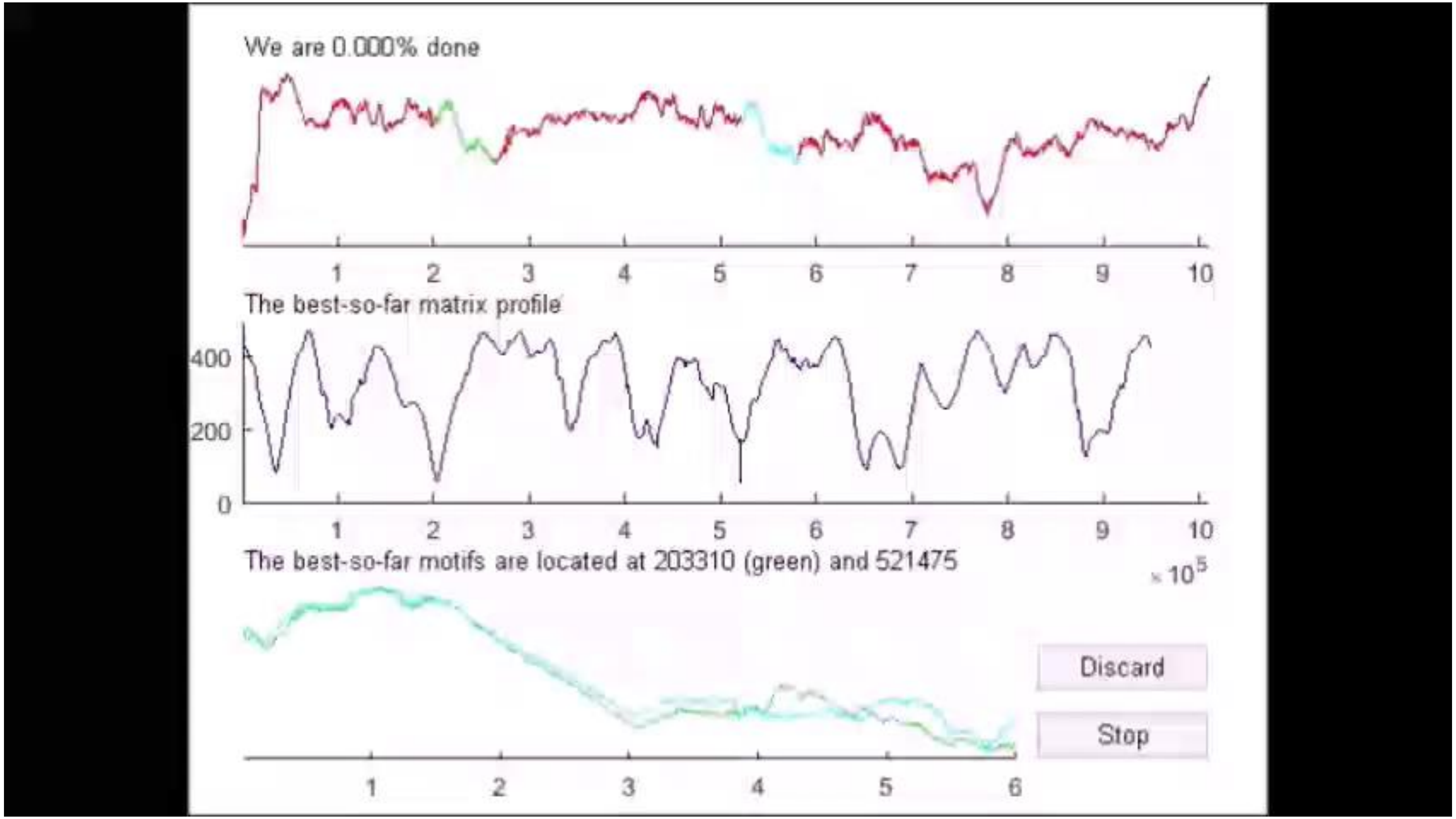
```
MP(1:n-m+1) = inf;
MPI(1:n-m+1) = -1;
for i = 1:n-m+1 in a random order
       d = MASS(T,T(i:i+m-1));
       d( max(i-m/4,1) : min(i+m/4-1,n-m+1))= NaN;
       [MP, ind] = min([MP ; d]);
       MPI(ind==2) = i;
end
```

Exploring the Anytime Property (1 of 3)



- The approximate matrix profile at 1,000 iteration is extremely similar to the exact solution.
- The convergence rate increases, for larger datasets.

Exploring the Anytime Property (2 of 3)



We are 0.000% done

The best-so-far matrix profile

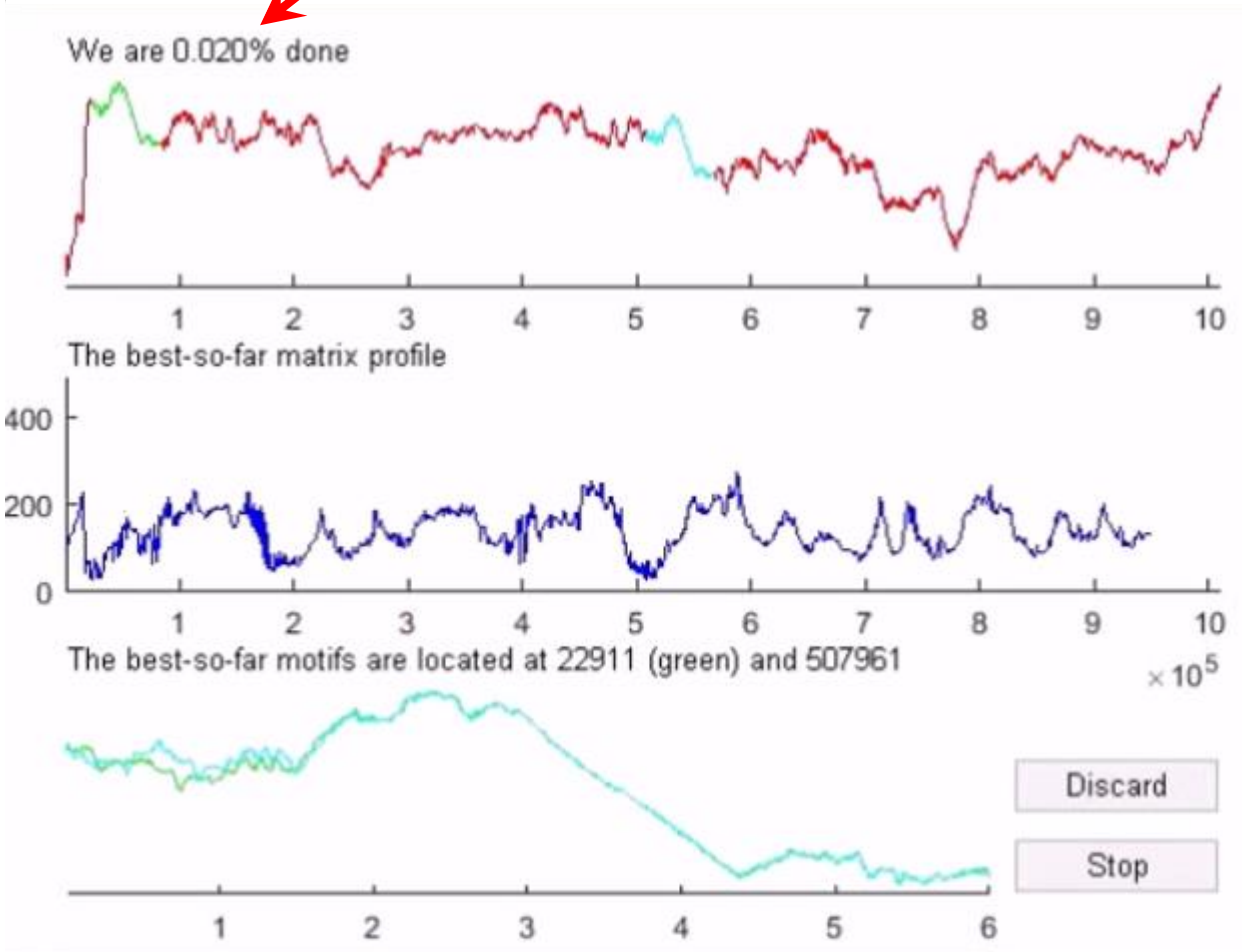
The best-so-far motifs are located at 203310 (green) and 521475

Play Video

Exploring the Anytime Property (3 of 3)

After doing only 1/500th of the computations, the basic shape of the MP has converged, and we have found the final correct motif.

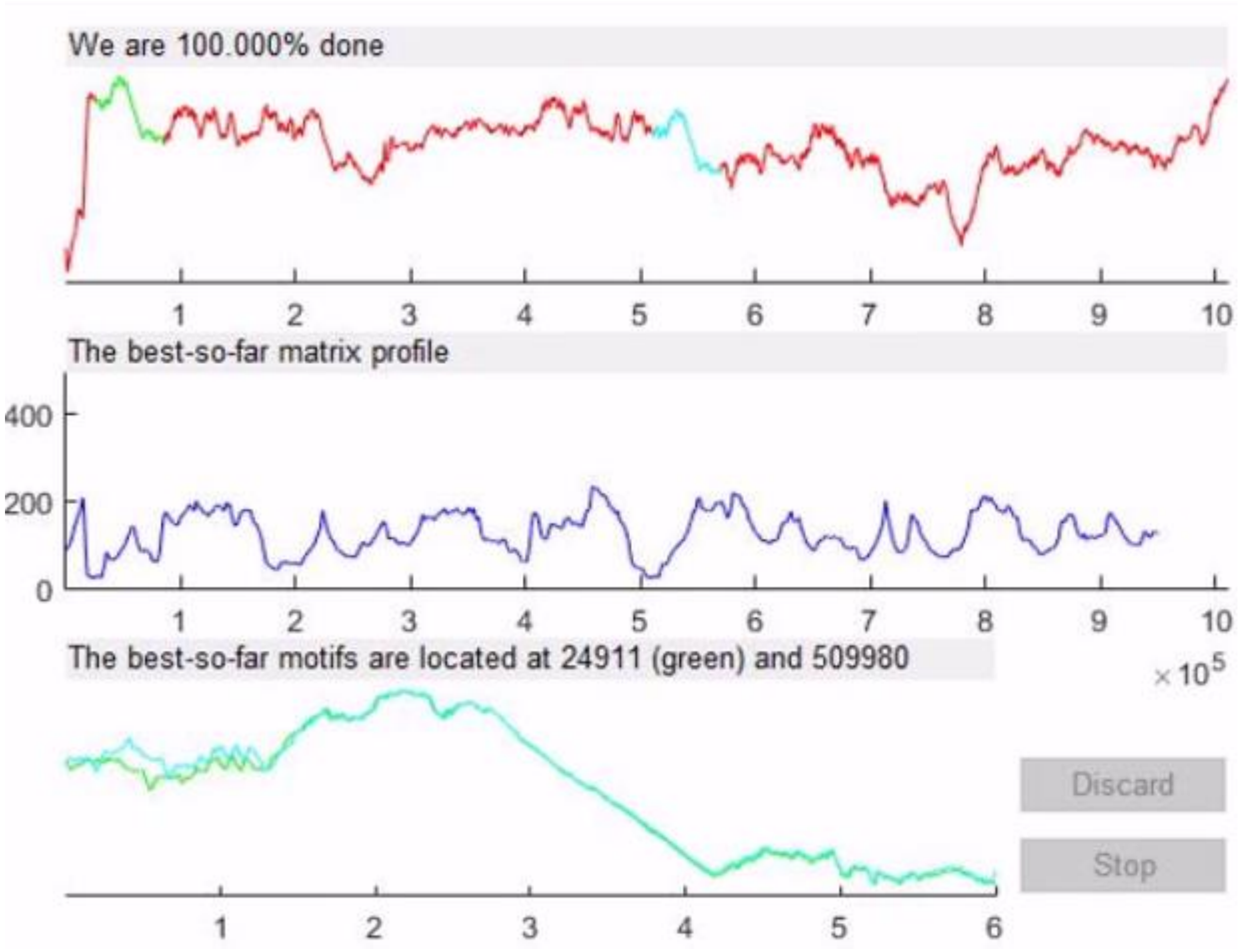
As an aside, note that the dimensionality of the subsequences here is 60,000.



We are 0.020% done

The best-so-far matrix profile

The best-so-far motifs are located at 22911 (green) and 507961



We are 100.000% done

The best-so-far matrix profile

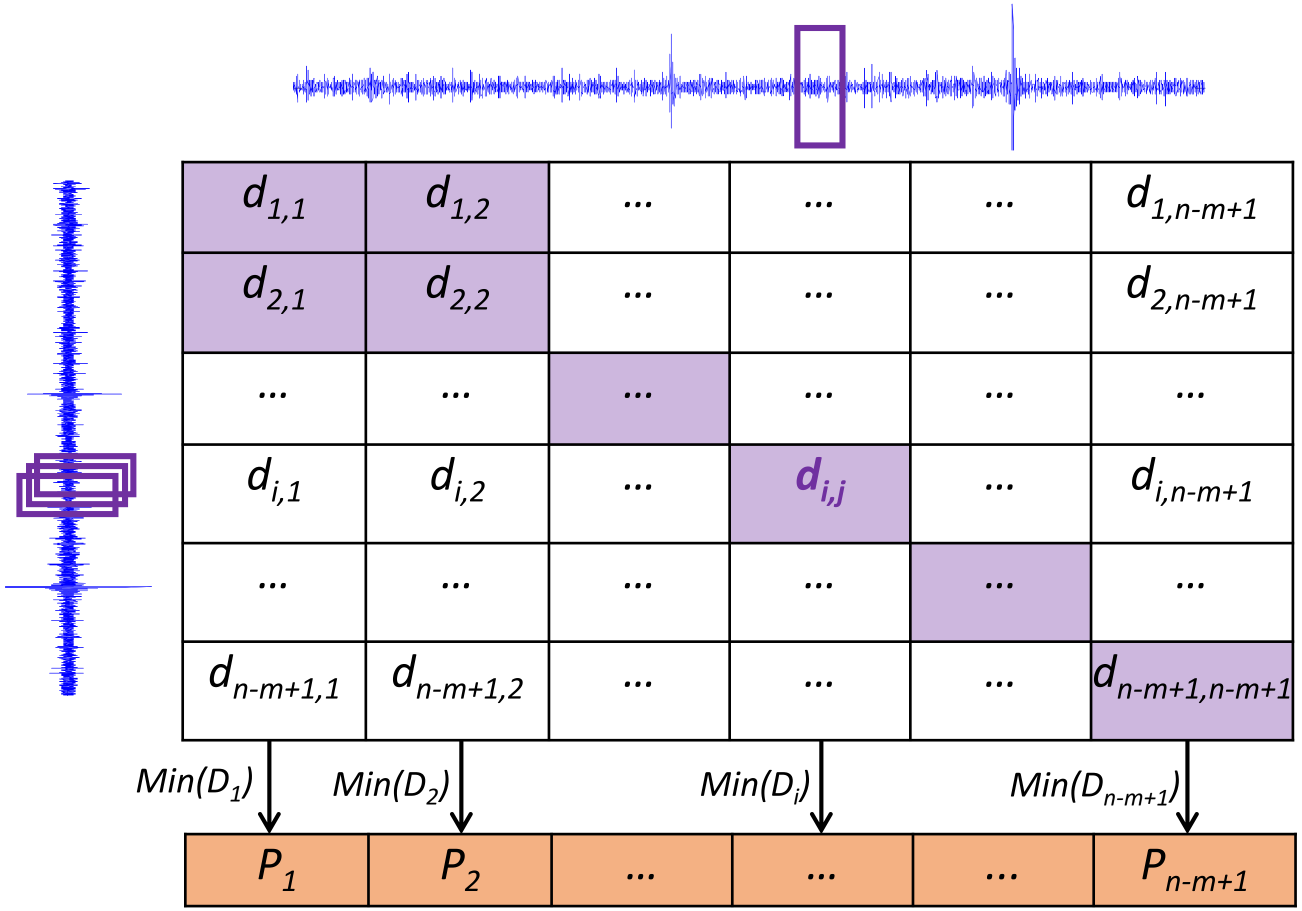
The best-so-far motifs are located at 24911 (green) and 509980

STOMP: Scalable Time series Ordered Matrix Profile

Can we improve STAMP?

STAMP computes distance profiles for rows in random order. Each distance profile is computed independently. However, the successive rows are profiles of two queries that overlap in $m-1$ observations.

We can exploit the overlap between successive queries while computing their distance profiles to build an $O(n^2)$ time, $O(n)$ space algorithm.



$d_{1,1}$	$d_{1,2}$	…	…	…	$d_{1,n-m+1}$
$d_{2,1}$	$d_{2,2}$	…	…	…	$d_{2,n-m+1}$
…	…	…	…	…	…
$d_{i,1}$	$d_{i,2}$	…	$d_{i,j}$	…	$d_{i,n-m+1}$
…	…	…	…	…	…
$d_{n-m+1,1}$	$d_{n-m+1,2}$	…	…	…	$d_{n-m+1,n-m+1}$

P_1	P_2	…	…	…	P_{n-m+1}

STOMP: Scalable Time series Ordered Matrix Profile

We have an ***O(n²)*** time, ***O(n)*** space algorithm called STOMP to evaluate it.

Recall our working formula:

$$T_iT_j = \sum_{k=0}^{m-1} t_{i+k}t_{j+k}$$

Dot product of the *i*th window and the *j*th window.

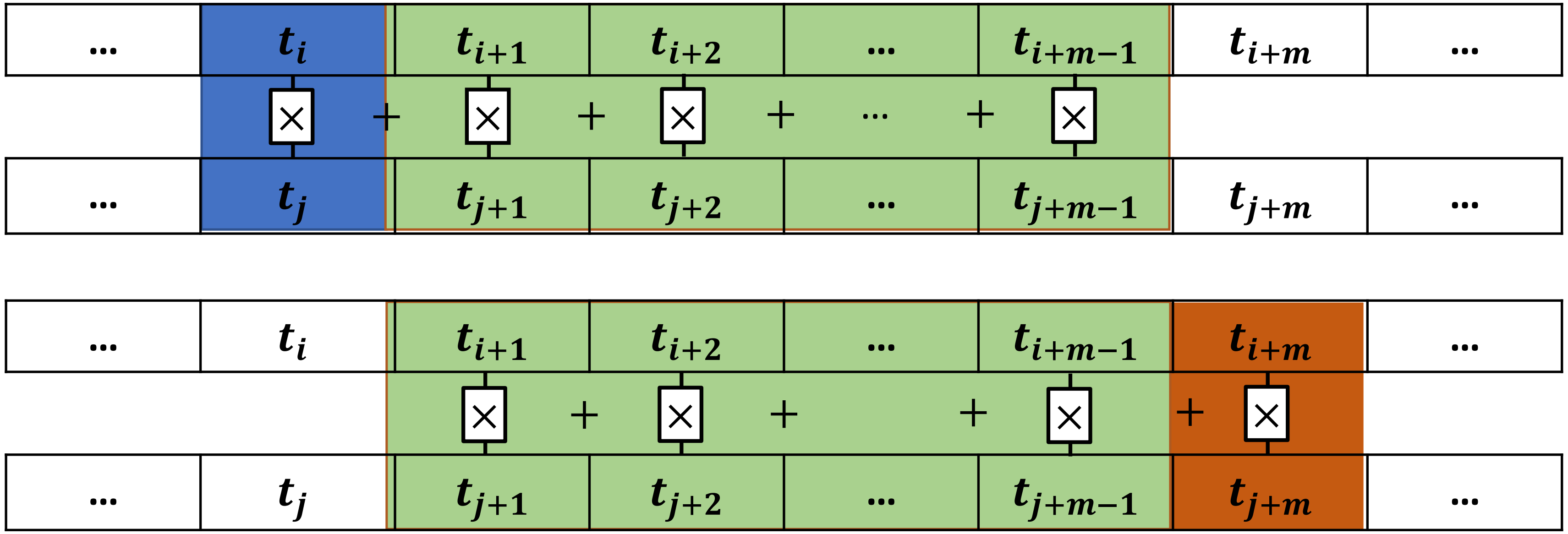
$$d_{i,j} = \sqrt{2m\left(1 - \frac{T_iT_j - m\mu_i\mu_j}{m\sigma_i\sigma_j}\right)}$$

- We precompute and store the means and standard deviations in *O(n)* space and time using the `movmean` and `movstd` functions.
- Once we know T_iT_j, it takes *O(1)* time to compute $d_{i,j}$

The relationship between T_iT_j and $T_{i+1}T_{j+1}$

$$T_iT_j = \sum_{k=0}^{m-1} t_{i+k}t_{j+k}$$

$$T_{i+1}T_{j+1} =$$



$$T_{i+1}T_{j+1} = T_iT_j - t_it_j + t_{i+m}t_{j+m}$$

$O(1)$ time complexity

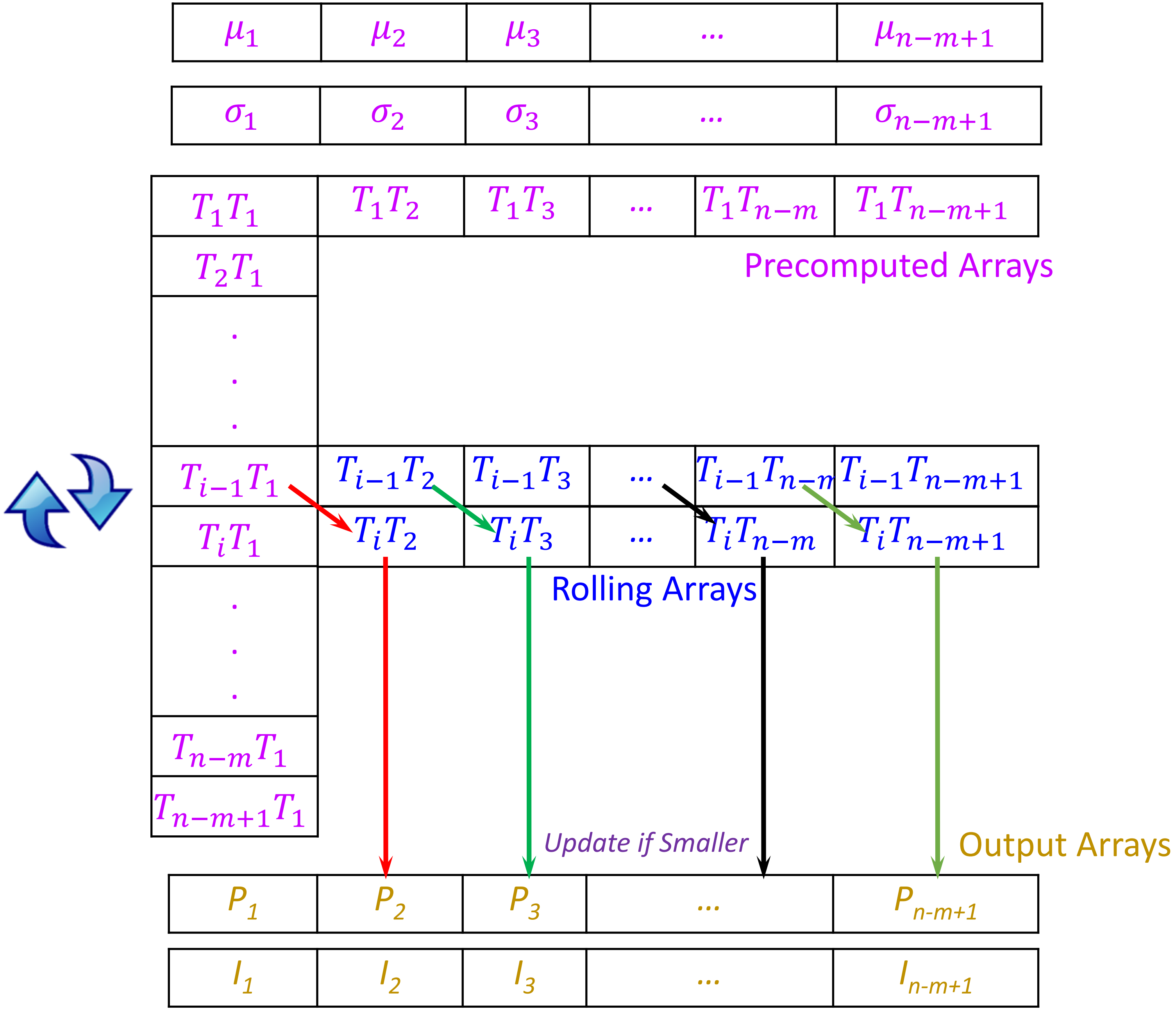
STOMP Algorithm: Computing the i^{th} Row

All means and standard deviations are precomputed. This costs linear time and space.

The first column and row of the matrix are identical and pre-computed by MASS.

We iterate through the rows. The previous row is maintained as a local array to feed dot products.

The dot products are converted to distance values and compared against the current best in the profile.



μ_1	μ_2	μ_3	…	μ_{n-m+1}

σ_1	σ_2	σ_3	…	σ_{n-m+1}

T_1T_1	T_1T_2	T_1T_3	…	T_1T_{n-m}	T_1T_{n-m+1}
T_2T_1					
. . .					
$T_{i-1}T_1$	$T_{i-1}T_2$	$T_{i-1}T_3$	…	$T_{i-1}T_{n-m}$	$T_{i-1}T_{n-m+1}$
T_iT_1	T_iT_2	T_iT_3	…	T_iT_{n-m}	T_iT_{n-m+1}
. . .					
$T_{n-m}T_1$					
$T_{n-m+1}T_1$					

Precomputed Arrays

Rolling Arrays

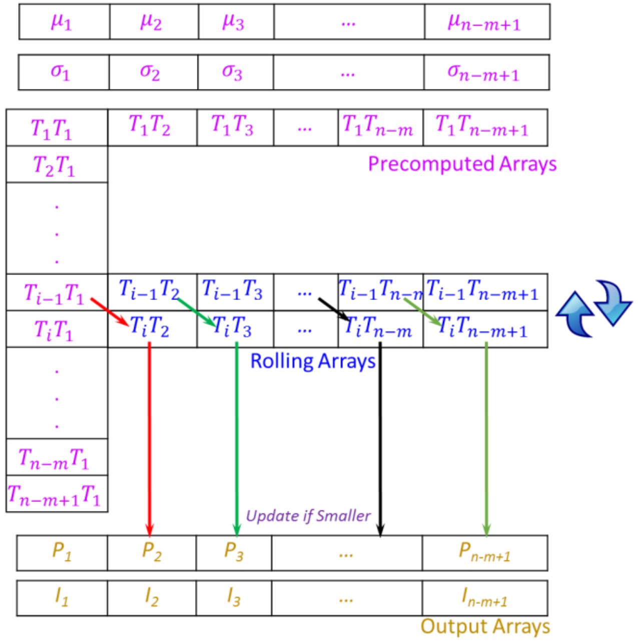
Update if Smaller

Output Arrays

P_1	P_2	P_3	…	P_{n-m+1}

I_1	I_2	I_3	…	I_{n-m+1}

SCRIMP: Scalable Column Independent Matrix Profile



μ_1 μ_2 μ_3 … μ_{n-m+1}

σ_1 σ_2 σ_3 … σ_{n-m+1}

T_1T_1 T_1T_2 T_1T_3 … T_1T_{n-m} T_1T_{n-m+1}

T_2T_1

Precomputed Arrays

$T_{i-1}T_1$ $T_{i-1}T_2$ $T_{i-1}T_3$ … $T_{i-1}T_{n-m}$ $T_{i-1}T_{n-m+1}$

T_iT_1 T_iT_2 T_iT_3 … T_iT_{n-m} T_iT_{n-m+1}

Rolling Arrays

$T_{n-m}T_1$

$T_{n-m+1}T_1$

Update if Smaller

P_1 P_2 P_3 … P_{n-m+1}

I_1 I_2 I_3 … I_{n-m+1}

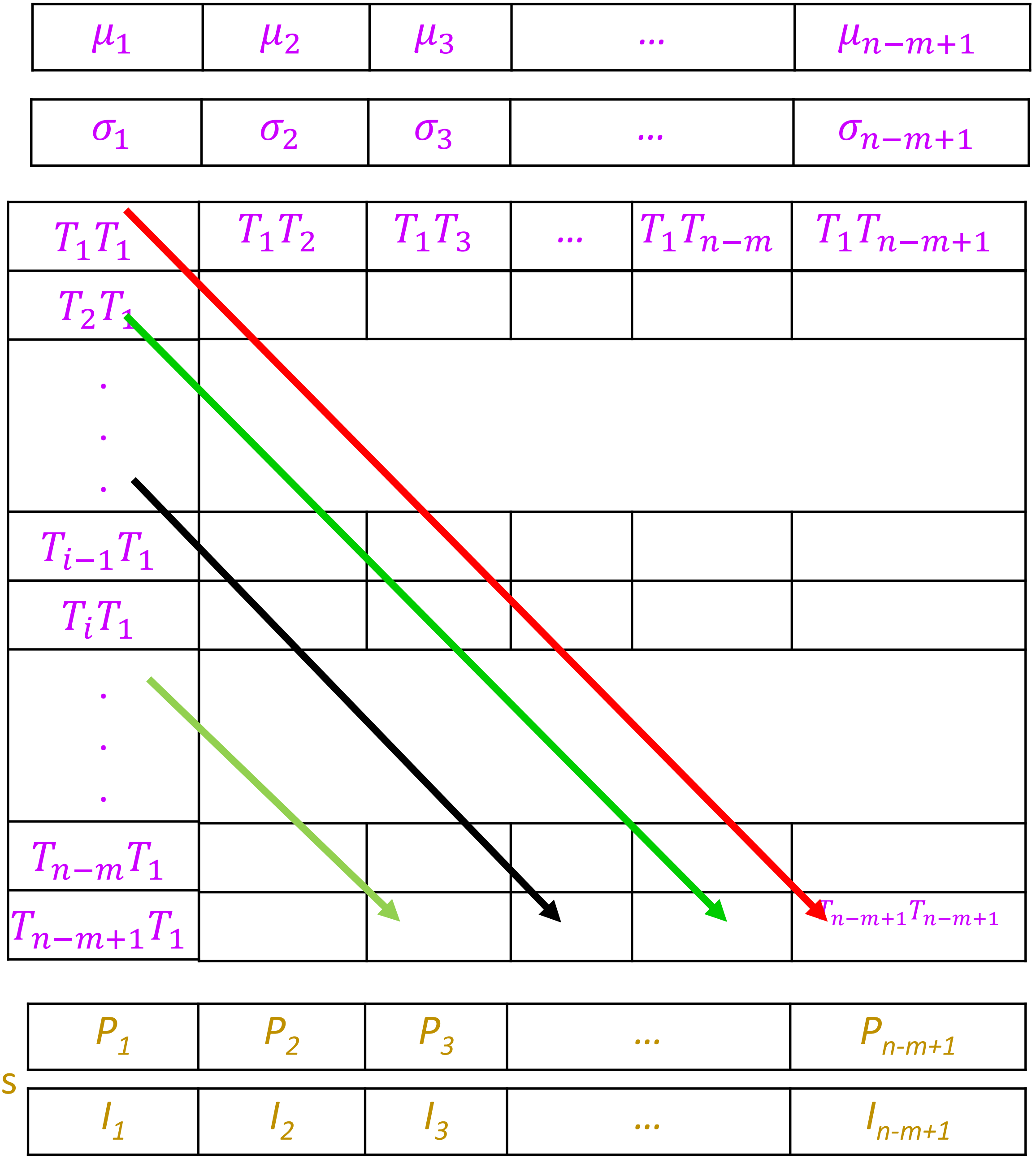
Output Arrays

STOMP iterates through rows. Each row depends on the previous row. Therefore random ordering is not suitable.

However, the diagonals are independent of each other. We exploit this property to randomize the computation so we achieve an anytime $O(n^2)$ algorithm.

We avoid computing the upper triangle of the matrix because the distance matrix is symmetric.

Precomputed Arrays



μ_1 μ_2 μ_3 … μ_{n-m+1}

σ_1 σ_2 σ_3 … σ_{n-m+1}

T_1T_1 T_1T_2 T_1T_3 … T_1T_{n-m} T_1T_{n-m+1}

T_2T_1

$T_{i-1}T_1$

T_iT_1

$T_{n-m}T_1$

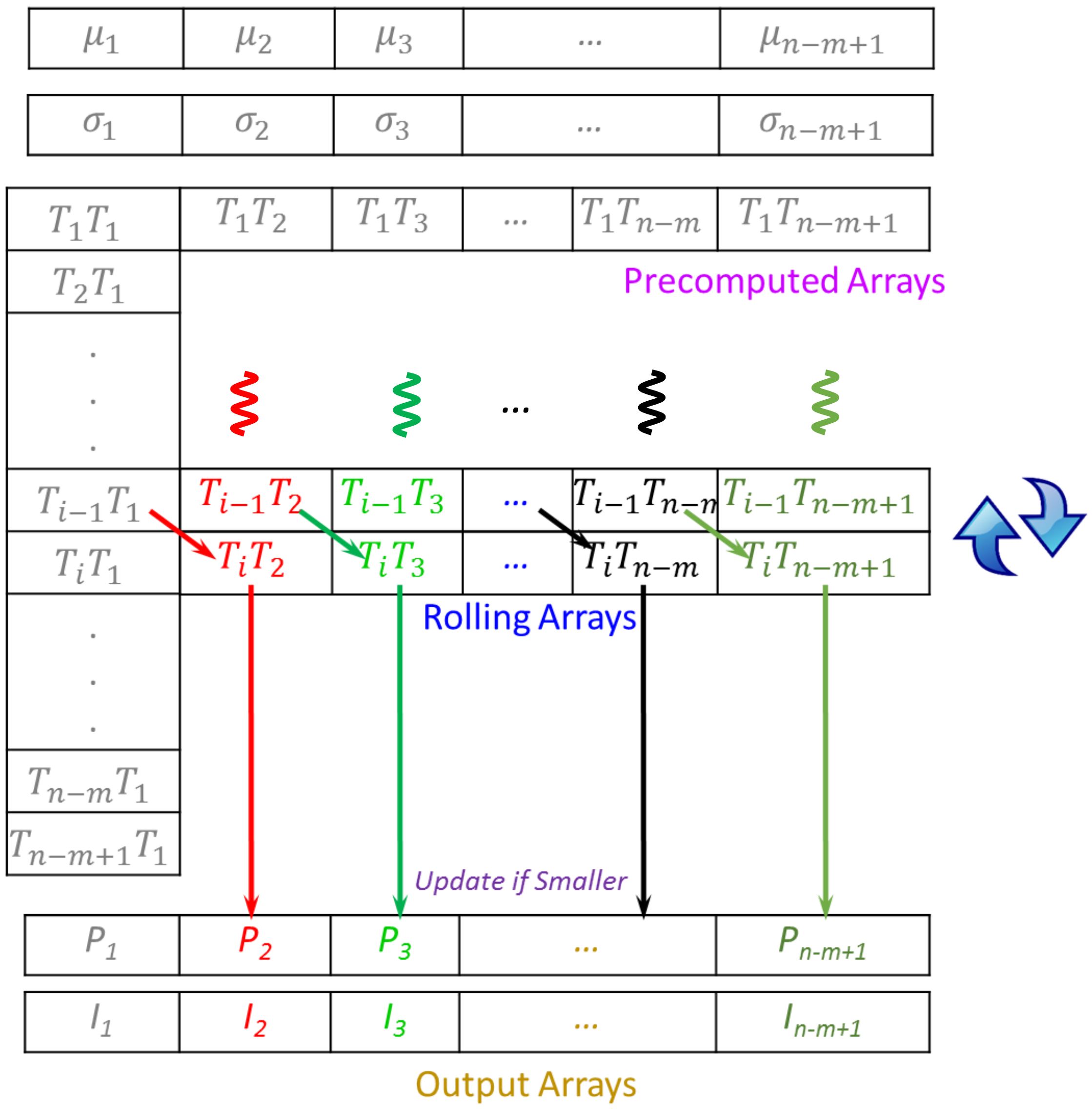
$T_{n-m+1}T_1$ $T_{n-m+1}T_{n-m+1}$

Output Arrays

P_1 P_2 P_3 … P_{n-m+1}

I_1 I_2 I_3 … I_{n-m+1}

Porting STOMP to GPU



μ_1	μ_2	μ_3	…	μ_{n-m+1}

σ_1	σ_2	σ_3	…	σ_{n-m+1}

T_1T_1	T_1T_2	T_1T_3	…	T_1T_{n-m}	T_1T_{n-m+1}

T_2T_1

.
.
.

Precomputed Arrays

…

$T_{i-1}T_1$	$T_{i-1}T_2$	$T_{i-1}T_3$	…	$T_{i-1}T_{n-m}$	$T_{i-1}T_{n-m+1}$
T_iT_1	T_iT_2	T_iT_3	…	T_iT_{n-m}	T_iT_{n-m+1}

Rolling Arrays

.
.
.

$T_{n-m}T_1$

$T_{n-m+1}T_1$

Update if Smaller

P_1	P_2	P_3	…	P_{n-m+1}
I_1	I_2	I_3	…	I_{n-m+1}

Output Arrays

Each thread is assigned to calculate one entry in the matrix profile in every iteration.

All the precomputed arrays are in global shared memory that is accessible to all threads.

Threads are synced after a row is processed to avoid race.

We can further optimize to compute only the lower triangle of the matrix, please see the paper.

Comparison of STAMP, STOMP and GPU-STOMP

For a fix subsequence length m=256: time

Algorithm ***n***	$\mathbf{2^{17}}$	$\mathbf{2^{18}}$	$\mathbf{2^{19}}$	$\mathbf{2^{20}}$
STAMP	15.1 min	1.17 hours	5.4 hours	24.4 hours
STOMP	4.21 min	0.3 hours	1.26 hours	5.22 hours
GPU-STOMP	10 sec	18 sec	46 sec	2.5 min

For large data, and **for the very first time in the literature, 100,000,000**

Algorithm ***m*** **|** ***n***	**2000 | 17,279,800**	**400 | 100,000,000**
STAMP (*estimated*)	*36.5 weeks*	*25.5 years*
STOMP (*estimated*)	*8.4 weeks*	*5.4 years*
GPU-STOMP	9.27 hours	12.13 days

Comparing the speed of STOMP with existing algorithms

For a time series of length 2^{18}: CPU time(memory usage)

Algorithm m	512	1,024	2,048	4,096
STOMP	501s (14MB)	506s (14MB)	490s (14MB)	490s (14MB)
Quick-Motif	27s (65MB)	151s (90MB)	630s (295MB)	695s (101MB)
MK	2040s (1.1GB)	N/A (>2GB)	N/A (>2GB)	N/A (>2GB)

Note: the time and space cost of STOMP is completely independent of any properties (noise, trends, stationarity etc.) of data.

Quick-Motif and MK are pruning based techniques with non-monotonic space need.

STOMP produces more information (i.e. Matrix Profile) while the others find the motifs only.

The Progress Bar Question: How long will it take for STOMP, SCRIMP to finish?

1. Given the inputs, the time is completely *deterministic*.
2. The time is completely independent of, *m* the length of the query. So long as *m* << *n.*
3. The time is completely independent of the structure/noise-level of data itself.

As it happens, these properties are *very* rare in data mining algorithms.

Can we use these properties to estimate time-to-finish?

The Progress Bar Question: How long will it take for STOMP, STAMP or SCRIMP to finish?

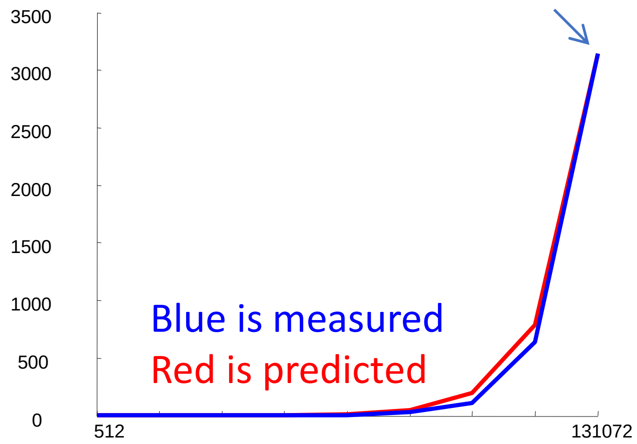
The time only depends on the length of the time series, *n*, and the hardware settings. To compute the time needed to finish, you just need to do *one* calibration run on any particular hardware configuration.

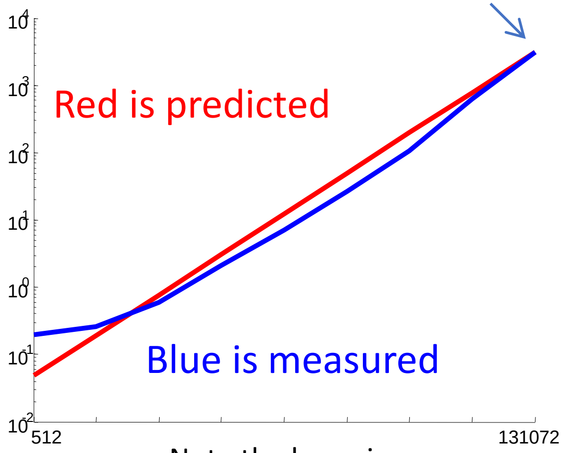
$$TimeNeeded = \frac{Time_{calibration}}{{n_{calibration}}^{2}} * {n_{new}}^{2}$$

How well does this work?

We *measured* the time needed for T = 512, 1024, 2048,… ,131072 (we used an old cheap laptop)

We then used the time measured for the 131,072 run, to predict the time needed for all the other runs. We plotted the two curves below. Note that the last point agrees by definition.





Note the log axis

Outline

Act 1

- Our Fundamental Assumption
- What is the (MP) Matrix Profile?
- Properties of the MP
- Developing a Visual Intuition for MP
- Basic Algorithms
  - MP Motif Discovery
  - MP Time Series Chains
  - MP Anomaly Discovery
  - MP Joins (self and AB)
  - MP Semantic Segmentation
- From Domain Agnostic to Domain Aware: The Annotation Vector (A simple way to use domain knowledge to adjust your results)
- The "*Matrix Profile and ten lines of code is all you need*" philosophy.
- Break

Act 2

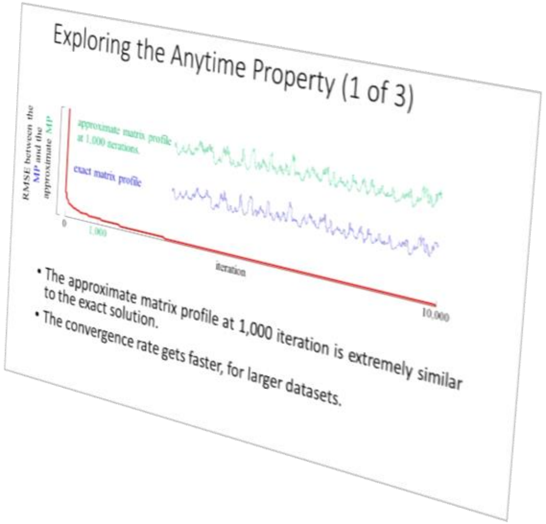
- Background on time series mining
  - Similarity Measures
  - Normalization
- Distance Profile
  - Brute Force Approach
  - Just-in-time Normalization
  - The MASS Algorithm
  - Extensions of MASS
- Matrix Profile
  - STAMP
  - STOMP
  - GPU-STOMP
  - SCRIMP
- Open problems to solve

Open Problems

- Find Lower Bounds
  - An $O(n)$ lower bound to distance profile
  - An $O(n \log n)$ lower bound to matrix profile
- Adapt to L1 distance
  - $O(n \log n)$ algorithm for distance profile
  - $O(n^2)$ algorithm for Matrix Profile
- Adapt to DTW distance
  - Can we create a Matrix Profile of 100M time series under warping?
- Variable Length Matrix Profile
  - Can we rank or sort neighbors of different lengths based on degree of fidelity?
- Scalability
  - Distributed Matrix Profile algorithm for horizontal scalability
  - Can we exploit hardware acceleration techniques for scalability?

Open Problems

- Domain Appropriate Matrix Profile
  - Recall that both STAMP and SCRIMP converge quickly with *random* ordering. However, could a *data-adaptive* ordering converge even faster?
  - We discussed the Annotation Vector (AV). While we can typically specify an AV with a few lines of code, can we *learn* a domain appropriate AV from user interaction?
- Visualization
  - At some scale, user-interfaces / user-experience become very important, we have largely glossed over this. Can we develop interactive visualization techniques for active exploration over MP?



Exploring the Anytime Property (1 of 3)

RMSE between the MP and the approximate MP
approximate matrix profile at 1,000 iterations.
exact matrix profile
0
1,000
iteration
10,000

- The approximate matrix profile at 1,000 iteration is extremely similar to the exact solution.
- The convergence rate gets faster, for larger datasets.

Questions?

The End!

KDD2017

Questions?

Visit the Matrix Profile Page

www.cs.ucr.edu/~eamonn/MatrixProfile.html

Visit the MASS Page

www.cs.unm.edu/~mueen/FastestSimilaritySearch.html

Please fill out an evaluation form, available in the back of the room.