Matrix Profile Tutorial by Hugh

1. Intuition
2. Application and Result
3. Calculation and Code
4. Algorithm to calculate

Naaek (Hugh) Chinpattanakarn

Github: https://github.com/hughnaaek

Mob: (+66) 834262415

Email: naaek.hugh@gmail.com

	Index	0	1	2	3	4	5	6	7
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]	[5,6]
0	[8,9]	3.2	2.0	3.2	2.0	8.1	9.2	6.4	4.2
1	[9,7]	4.1	1.0	2.2	3.6	6.0	8.9	6.3	4.1
2	[7,5]	3.6	2.2	1.0	4.1	4.5	6.3	4.0	2.2
3	[5,6]	2.0	3.2	2.0	3.2	6.4	5.0	2.2	0.0
Matrix Profile		2.0	1.0	1.0	2.0	4.5	5.0	2.2	0.0
Location array		3	1	2	2	4.5	5.0	2.2	0.0

Matrix Profile: Intuition

One Sentence: Given two time series, which part of the first time series is the most similar to a part of the second time series?

Caution: Matrix Profile(MP) is just one dimensional array from an algorithm.
In this slides, algorithm(two ts) = (MP,location array) is referred as Matrix Profile's algorithm.

Also, the output of MP algorithm (**MP index**) referred as **location array**

Example:
Let T1 is **reference TS** for **query origin TS** (T2) to calculate distance.

T1(ref TS) : [5,8,7,6,9,1,3,5,6]
T2(query origin TS) : [8,9,7,5,6]

The only parameter of the algorithm

1. Slicing window

example window size(m) = 2
T1 : [5,8],[8,7],[7,6],[6,9],[9,1],[1,3],[3,5],[5,6]
T2 : [8,9],[9,7],[7,5],[5,6]

2. Similarity search

MP func uses euclidean distance function with z-normalization of each partition. Normalizing for compare the shape of time series, the size to compare must be in the same scale.

General question that Matrix Profile function solves:
Which part of T2(query) is the most similar to a part of T1?

Calculating direction

	Index	0	1	2	3	4	5	6	7
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]	[5,6]
0	[8,9]	3.2	2.0	3.2	2.0	8.1	9.2	6.4	4.2
1	[9,7]	4.1	1.0	2.2	3.6	6.0	8.9	6.3	4.1
2	[7,5]	3.6	2.2	1.0	4.1	4.5	6.3	4.0	2.2
3	[5,6]	2.0	3.2	2.0	3.2	6.4	5.0	2.2	0.0
Matrix Profile		2.0	1.0	1.0	2.0	4.5	5.0	2.2	0.0
Location array		3	1	2	2	4.5	5.0	2.2	0.0

How to interpret MP product generally?

The **i th partition** of **ref TS** has its **closed distance** to the **location_arr[i] th query TS** with z-normalized euclidean **distance of MP[i]**.

General Calculation

*The matrix above and the product(MP) result in name Matrix Profile respectively.

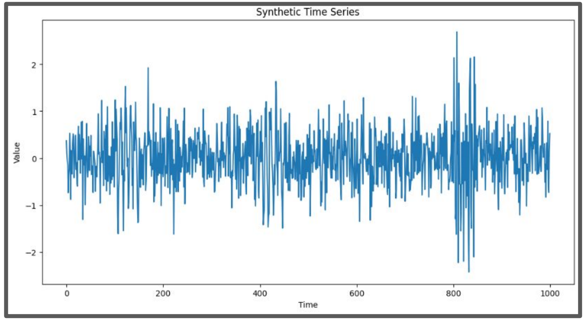
*For this example, use result of euclidean distance instead due to its obvious result to distinguish the lowest value.

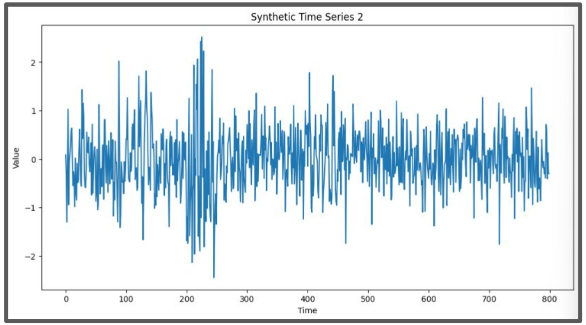
Matrix Profile: Application & Result

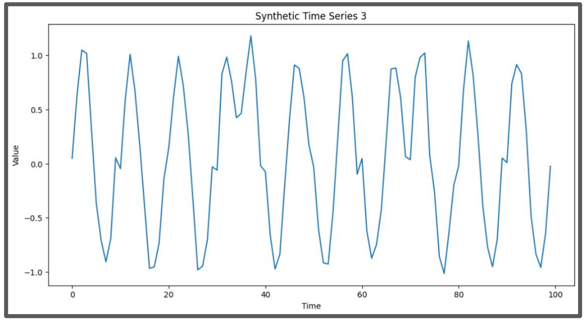
Main Application	MP func's product used?	Condition: Query and Ref TS
Motif Discovery	Matrix Profile array	same
Discord Detection	Matrix Profile array	same
TS Segmentation	Location array	same
Two TS Similarity Search	Matrix Profile array	No

Generating Factor	TS 1	TS 2	TS 3	Unit
Length	1000	800	100	Timepoint
Motif_interval	100,400	100,400	0,100	Timepoint
Motif_length	50	50	10	Timepoint
Discord_interval	200,250	200,250	35,70	Timepoint
Discord_length	50	50	35	Timepoint
Noice	Normal Dist.with std = 0.5 (TS 3 = 0.1)			

Time Series as Input







Application Example in this slides

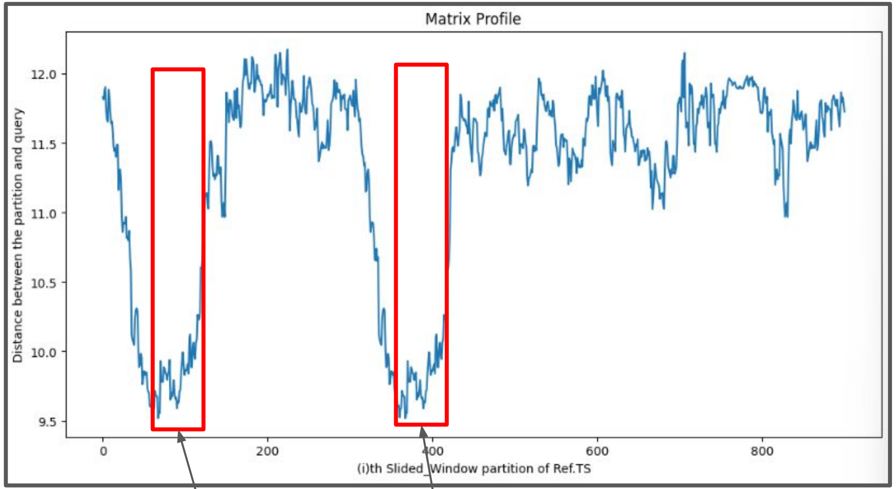
Synthetic TS 1:
- Motif Discovery
- TS Segmentation
- Two Ts similarity search

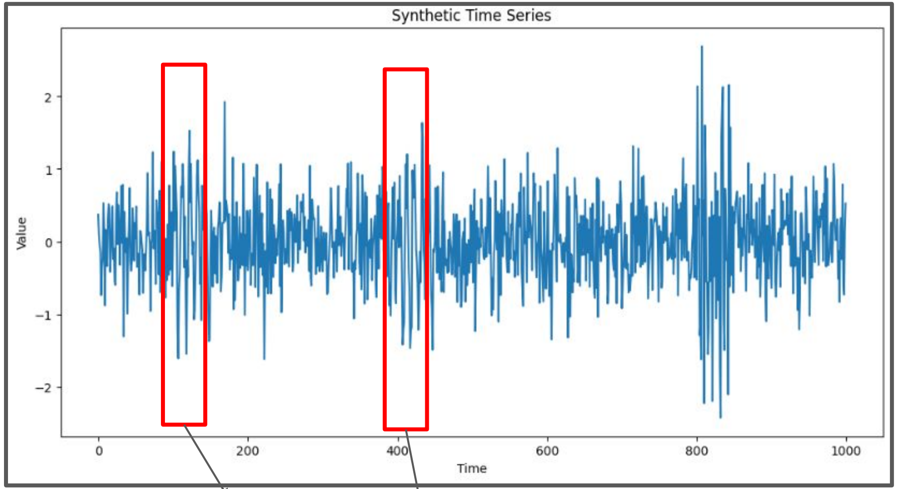
Synthetic TS 2:
- Two Ts similarity search

Synthetic TS 3:
- Discord Detection
- Algorithm speed Testing

Matrix Profile: Application & Result (Motif Discovery)

sliding window (m) = 200





When the **nearest non-trivial query comparing.** it causes the **lowest distance value**.

The similarity of synthetic **motif patterns** is **not obvious** enough due to noise.

Because of **similarity** of the **two parts** in the time serie

Ground Truth Table

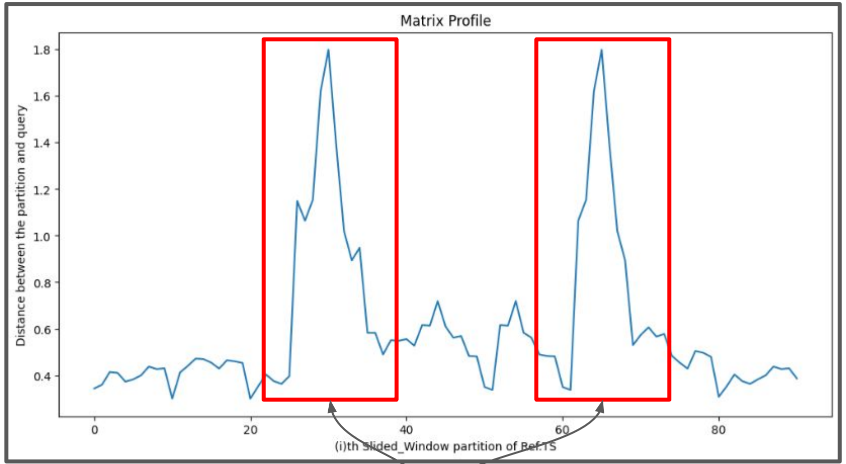
Generating Factor	TS 1
Length	1000
Motif_interval	100,400
Motif_length	50

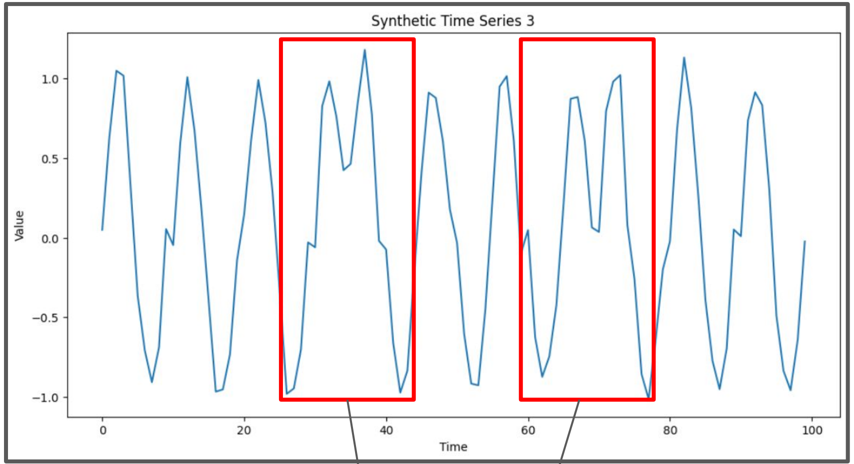
When **comparing the result with Ground Truth**, it effectively provide result of discord (**anomaly** in the time serie)

To find **location** of motif,
It is from **lowest distance value's index** of MP array(mpi) **to mpi + m** on the time serie.

Matrix Profile: Application & Result (Discord Detection)

sliding window (m) = 10





When the **nearest non-trivial query comparing.** it still causes the **highest distance value**.

The **obviously different patterns** in the time serie

Ground Truth Table

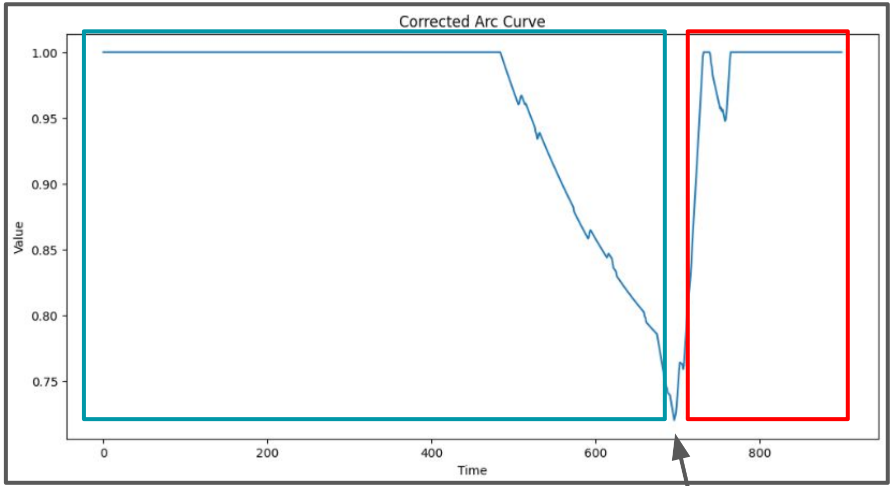
Discord_Interval	35,70 (timepoint)
Discord_length	35 (timepoint)

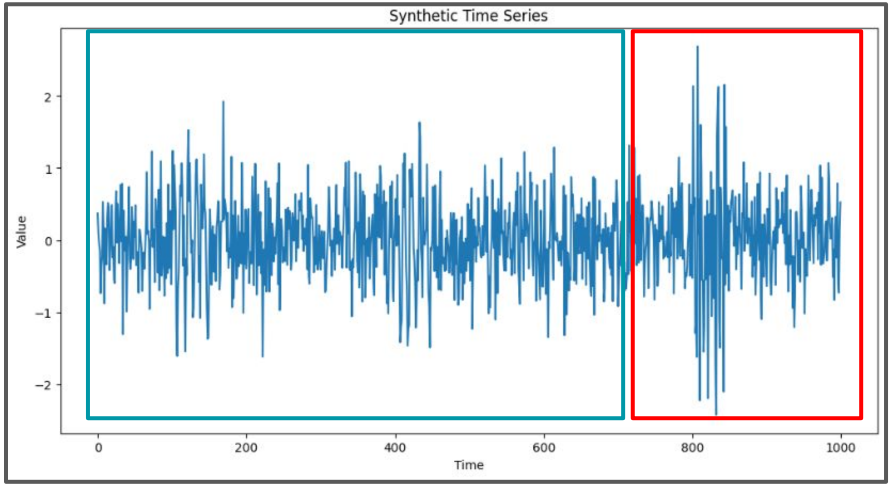
When **comparing the result with Ground Truth**, it effectively provide result of discord (**anomaly** in the time serie)

The only **problem** of using this tool is **size of the sliding window** that effect **obviousness** of the result.
Magic num = 10%

Matrix Profile: Application & Result (TS Segmentation)

sliding window (m) = 200



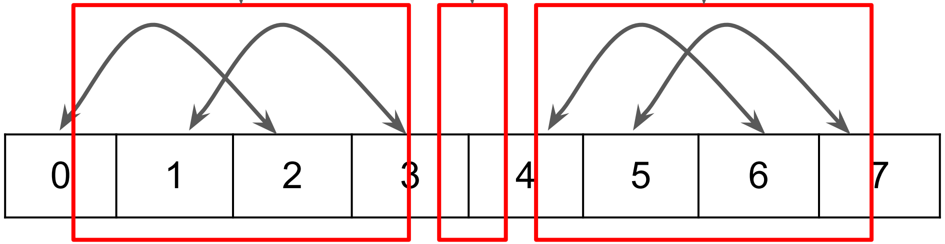


696th timepoint with the **lowest** value 0.719 is the point of **changing regime**

Highest value of corrected arc curve area

Lowest value of corrected arc curve area

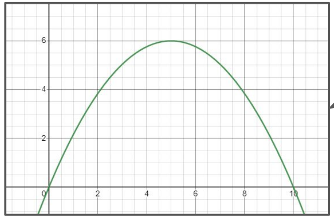
Highest value of corrected arc curve area



"Arc-dense area indicates **similarity** in the area."

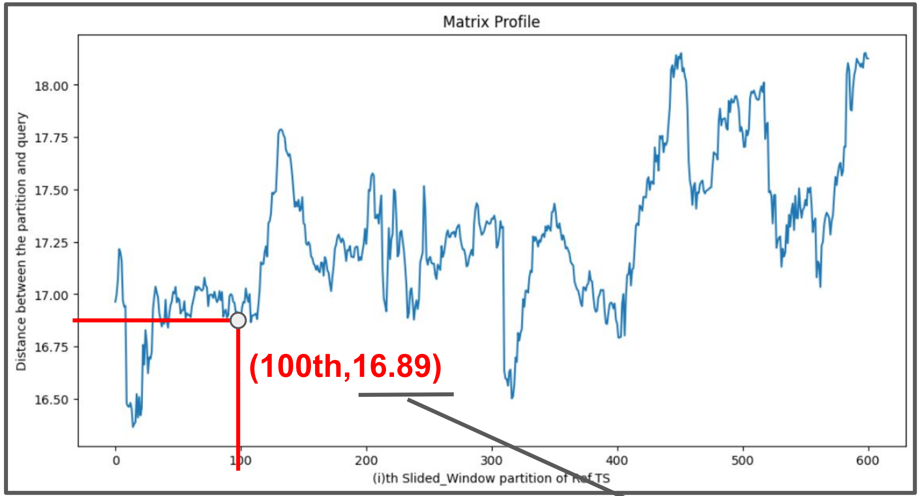
Graph of Ideal Arc Curve Function to calculate the Value

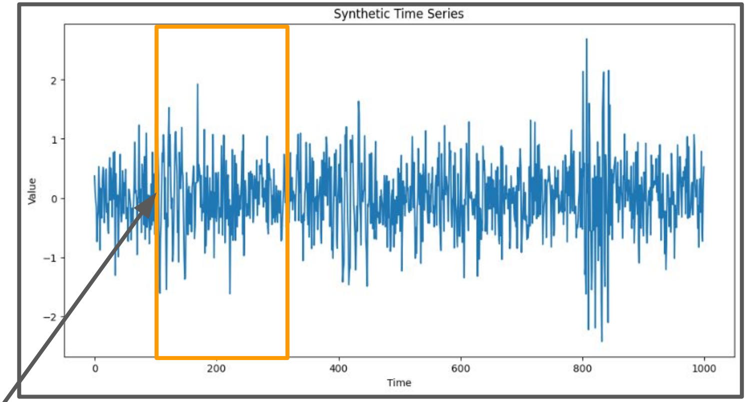
The **Transition** is the between area of arc-dense or the similar where there are **fewest arc value**.



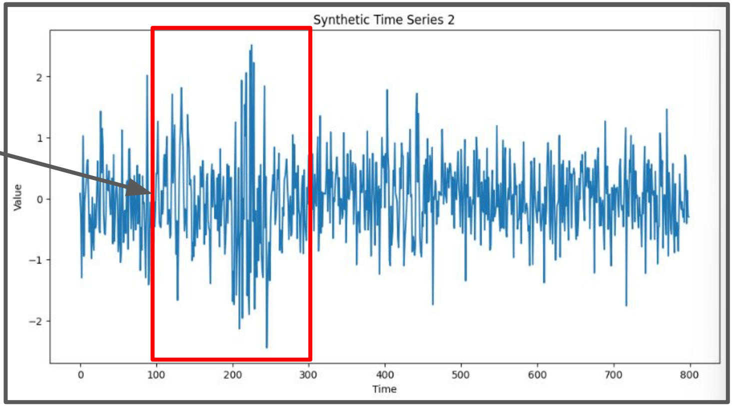
Matrix Profile: Application & Result (Two TS Similarity Search)

sliding window (m) = 200





Query Time Series



Ref. Time Series

Distance between window starts at 100th of Ref ts and window starts at (location_array[100]) th of Query ts

The **i th partition** of **ref TS** has its **closed distance** to the **location_arr[i] th query TS** with z-normalized euclidean **distance of MP[i]**.

If we have matrix profile array and location array,
We know 1. The closest partition pair of Ref ts and Query ts
2. Where the partitions are in both ts
3. Distance of both partition in z-normalized euclidean distance

Matrix Profile: Calculation & Code

1. Set initial value of Matrix Profile array and Location array.

	Index	0	1	2	3	4	5	6	7
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]	[5,6]
0	[8,9]								
1	[9,7]								
2	[7,5]								
3	[5,6]								
Matrix Profile		**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**
Location array		**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**

```
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

Reason of initialing inf: comparing with distance profiles to update the lower value.

Matrix Profile: Calculation & Code

2. Sliding window the query time series to have a set of query.

	Index	0	1	2	3	4	5	6	7
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]	[5,6]
0	[8,9]								
1	[9,7]								
2	[7,5]								
3	[5,6]								
Matrix Profile		**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**
Location array		**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**

```
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

Matrix Profile: Calculation & Code

3. Calculate Distance profile(distance between each partition of reference time series and the query) by sliding the query throughout the reference time series .

	Index	0	1	2	3	4	5	6	7
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]	[5,6]
0	[8,9]	**3.2**	**2.0**	**3.2**	**2.0**	**8.1**	**9.2**	**6.4**	**4.2**
1	[9,7]								
2	[7,5]								
3	[5,6]								
Matrix Profile		**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**
Location array		**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**

```
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

Note: Euclidean distance is used instead of Z-normalization euclidean distance due to providing more obvious result to compare the lowest value for the example.

“Normalization is a must for compare the shape of time series, the size to compare must be in the same scale”

Matrix Profile: Calculation & Code

4. Comparing Distance profile with matrix profile and update location array for the lower distance.

	Index	0	1	2	3	4	5	6	7
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]	[5,6]
0	[8,9]	**3.2**	**2.0**	**3.2**	**2.0**	**8.1**	**9.2**	**6.4**	**4.2**
1	[9,7]								
2	[7,5]								
3	[5,6]								
Matrix Profile		**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**
Location array		**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

```python
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

Matrix Profile: Calculation & Code

5. Update matrix profile for the lower distance of distance profile.

	Index	0	1	2	3	4	5	6	7
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]	[5,6]
0	[8,9]	**3.2**	**2.0**	**3.2**	**2.0**	**8.1**	**9.2**	**6.4**	**4.2**
1	[9,7]								
2	[7,5]								
3	[5,6]								
Matrix Profile		**3.2**	**2.0**	**3.2**	**2.0**	**8.1**	**9.2**	**6.4**	**4.2**
Location array		**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

```python
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

Matrix Profile: Calculation & Code

6. Calculate Distance profile of the next query.

	Index	0	1	2	3	4	5	6	7
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]	[5,6]
0	[8,9]	**3.2**	**2.0**	**3.2**	**2.0**	**8.1**	**9.2**	**6.4**	**4.2**
1	[9,7]	**4.1**	**1.0**	**2.2**	**3.6**	**6.0**	**8.9**	**6.3**	**4.1**
2	[7,5]								
3	[5,6]								
Matrix Profile		**3.2**	**2.0**	**3.2**	**2.0**	**8.1**	**9.2**	**6.4**	**4.2**
Location array		**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

```
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

“Normalization is a must for compare the shape of time series, the size to compare must be in the same scale”

Note: Euclidean distance used instead of Z-normalization euclidean distance due to providing more obvious result to compare the lowest value for the example.

Matrix Profile: Calculation & Code

7. Comparing Distance profile with matrix profile and update location array for the lower distance.

	Index	0	1	2	3	4	5	6	7
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]	[5,6]
0	[8,9]	3.2	2.0	3.2	2.0	8.1	9.2	6.4	4.2
1	[9,7]	4.1	1.0	2.2	3.6	6.0	8.9	6.3	4.1
2	[7,5]								
3	[5,6]								
Matrix Profile		3.2	2.0	3.2	2.0	8.1	9.2	6.4	4.2
Location array		0	0	0	0	0	0	0	0

```python
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

Matrix Profile: Calculation & Code

8. Comparing Distance profile with matrix profile and update location array for the lower distance.

	Index	0	1	2	3	4	5	6	7
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]	[5,6]
0	[8,9]	3.2	2.0	3.2	2.0	8.1	9.2	6.4	4.2
1	[9,7]	4.1	1.0	2.2	3.6	6.0	8.9	6.3	4.1
2	[7,5]								
3	[5,6]								
Matrix Profile		3.2	2.0	3.2	2.0	8.1	9.2	6.4	4.2
Location array		0	1	1	0	1	1	1	1

```
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

Matrix Profile: Calculation & Code

9. Update matrix profile for the lower distance of distance profile.

	Index	0	1	2	3	4	5	6	7
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]	[5,6]
0	[8,9]	**3.2**	**2.0**	**3.2**	**2.0**	**8.1**	**9.2**	**6.4**	**4.2**
1	[9,7]	**4.1**	**1.0**	**2.2**	**3.6**	**6.0**	**8.9**	**6.3**	**4.1**
2	[7,5]								
3	[5,6]								
Matrix Profile		**3.2**	**1.0**	**2.2**	**2.0**	**6.0**	**8.9**	**6.3**	**4.1**
Location array		**0**	**1**	**1**	**0**	**1**	**1**	**1**	**1**

```python
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

Matrix Profile: Calculation & Code

10. Repeat it until having exact result.

	Index	0	1	2	3	4	5	6	7
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]	[5,6]
0	[8,9]	**3.2**	**2.0**	**3.2**	**2.0**	**8.1**	**9.2**	**6.4**	**4.2**
1	[9,7]	**4.1**	**1.0**	**2.2**	**3.6**	**6.0**	**8.9**	**6.3**	**4.1**
2	[7,5]	**3.6**	**2.2**	**1.0**	**4.1**	**4.5**	**6.3**	**4.0**	**2.2**
3	[5,6]	**2.0**	**3.2**	**2.0**	**3.2**	**6.4**	**5.0**	**2.2**	**0.0**
Matrix Profile		**2.0**	**1.0**	**1.0**	**2.0**	**4.5**	**5.0**	**2.2**	**0.0**
Location array		**3**	**1**	**2**	**2**	**4.5**	**5.0**	**2.2**	**0.0**

```
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

Note: If it is still repeating calculated, the matrix profile and location array are still **approximated result that converge to the exact result**.

Matrix Profile: Calculation & Code (QueryTs and RefTs Same)

	Index	0	1	2	3	4	5	6
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]
0	[5,8]							
1	[8,7]							
2	[7,6]							
3	[6,9]							
4	[9,1]							
5	[1,3]							
6	[3,5]							
Matrix Profile		**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**
Location array		**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**

1. Set initial value of Matrix Profile array and Location array.

```
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

Reason of initialing inf: comparing with distance profiles to update the lower value.

Matrix Profile: Calculation & Code (QueryTs and RefTs Same)

	Index	0	1	2	3	4	5	6
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]
0	[5,8]	**0.0**	**3.2**	**2.8**	**1.4**	**8.1**	**6.4**	**3.6**
1	[8,7]							
2	[7,6]							
3	[6,9]							
4	[9,1]							
5	[1,3]							
6	[3,5]							
Matrix Profile		**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**
Location array		**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**

2. Calculate Distance profile(distance between each partition and the query).

```
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

Note: Euclidean distance used instead of Z-normalization euclidean distance due to providing more obvious result to compare the lowest value for the example.

Matrix Profile: Calculation & Code (QueryTs and RefTs Same)

3. Trivial Match provide non-needed insight.

	Index	0	1	2	3	4	5	6
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]
0	[5,8]	**0.0**	**3.2**	**2.8**	**1.4**	**8.1**	**6.4**	**3.6**
1	[8,7]							
2	[7,6]							
3	[6,9]							
4	[9,1]							
5	[1,3]							
6	[3,5]							
Matrix Profile		**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**
Location array		**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**

```
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

```
dp = np.array(dp)
if np.array_equal(tsA, tsB): dp[idx] = np.inf
```

Matrix Profile: Calculation & Code (QueryTs and RefTs Same)

	Index	0	1	2	3	4	5	6
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]
0	[5,8]	**inf**	**3.2**	**2.8**	**1.4**	**8.1**	**6.4**	**3.6**
1	[8,7]							
2	[7,6]							
3	[6,9]							
4	[9,1]							
5	[1,3]							
6	[3,5]							
Matrix Profile		**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**
Location array		**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**

4. Setting Trivial Match to inf (exclusive zone).

```
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

```
dp = np.array(dp)
if np.array_equal(tsA, tsB): dp[idx] = np.inf
```

Matrix Profile: Calculation & Code (QueryTs and RefTs Same)

	Index	0	1	2	3	4	5	6
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]
0	[5,8]	**inf**	**3.2**	**2.8**	**1.4**	**8.1**	**6.4**	**3.6**
1	[8,7]							
2	[7,6]							
3	[6,9]							
4	[9,1]							
5	[1,3]							
6	[3,5]							
Matrix Profile		**inf**	**inf**	**inf**	**inf**	**inf**	**inf**	**inf**
Location array		**inf**	**1**	**2**	**3**	**4**	**5**	**6**

5. Comparing Distance profile with matrix profile and update location array for the lower distance.

```
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

Matrix Profile: Calculation & Code (QueryTs and RefTs Same)

	Index	0	1	2	3	4	5	6
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]
0	[5,8]	**inf**	**3.2**	**2.8**	**1.4**	**8.1**	**6.4**	**3.6**
1	[8,7]							
2	[7,6]							
3	[6,9]							
4	[9,1]							
5	[1,3]							
6	[3,5]							
Matrix Profile		**inf**	**3.2**	**2.8**	**1.4**	**8.1**	**6.4**	**3.6**
Location array		**inf**	**1**	**2**	**3**	**4**	**5**	**6**

6. Update matrix profile for the lower distance of distance profile.

```
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

Matrix Profile: Calculation & Code (QueryTs and RefTs Same)

	Index	0	1	2	3	4	5	6
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]
0	[5,8]	**0.0**	**3.2**	**2.8**	**1.4**	**8.1**	**6.4**	**3.6**
1	[8,7]	**3.2**	**0.0**	**1.4**	**2.8**	**6.1**	**8.1**	**5.4**
2	[7,6]							
3	[6,9]							
4	[9,1]							
5	[1,3]							
6	[3,5]							
Matrix Profile		**inf**	**3.2**	**2.8**	**1.4**	**8.1**	**6.4**	**3.6**
Location array		**inf**	**1**	**2**	**3**	**4**	**5**	**6**

Trivial Match

7. Calculate Distance profile

```
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

Note: Euclidean distance used instead of Z-normalization euclidean distance due to providing more obvious result to compare the lowest value for the example.

Matrix Profile: Calculation & Code (QueryTs and RefTs Same)

	Index	0	1	2	3	4	5	6
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]
0	[5,8]	**0.0**	**3.2**	**2.8**	**1.4**	**8.1**	**6.4**	**3.6**
1	[8,7]	**3.2**	**inf**	**1.4**	**2.8**	**6.1**	**8.1**	**5.4**
2	[7,6]							
3	[6,9]							
4	[9,1]							
5	[1,3]							
6	[3,5]							
Matrix Profile		**inf**	**3.2**	**2.8**	**1.4**	**8.1**	**6.4**	**3.6**
Location array		**inf**	**1**	**2**	**3**	**4**	**5**	**6**

8. Comparing Distance profile with matrix profile and update location array for the lower distance.

```
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

Matrix Profile: Calculation & Code (QueryTs and RefTs Same)

	Index	0	1	2	3	4	5	6
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]
0	[5,8]	**0.0**	**3.2**	**2.8**	**1.4**	**8.1**	**6.4**	**3.6**
1	[8,7]	**3.2**	**inf**	**1.4**	**2.8**	**6.1**	**8.1**	**5.4**
2	[7,6]							
3	[6,9]							
4	[9,1]							
5	[1,3]							
6	[3,5]							
Matrix Profile		**inf**	**3.2**	**2.8**	**1.4**	**8.1**	**6.4**	**3.6**
Location array		**inf**	**1**	**2**	**3**	**4**	**5**	**6**

9. Setting Trivial Match to inf (exclusive zone).

```
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

```
dp = np.array(dp)
if np.array_equal(tsA, tsB): dp[idx] = np.inf
```

Matrix Profile: Calculation & Code (QueryTs and RefTs Same)

	Index	0	1	2	3	4	5	6
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]
0	[5,8]	**0.0**	**3.2**	**2.8**	**1.4**	**8.1**	**6.4**	**3.6**
1	[8,7]	**3.2**	**inf**	**1.4**	**2.8**	**6.1**	**8.1**	**5.4**
2	[7,6]							
3	[6,9]							
4	[9,1]							
5	[1,3]							
6	[3,5]							
Matrix Profile		**3.2**	**3.2**	**1.4**	**1.4**	**6.1**	**6.4**	**3.6**
Location array		**1**	**0**	**1**	**0**	**1**	**0**	**0**

10. Update matrix profile for the lower distance of distance profile.

```
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

Matrix Profile: Calculation & Code (QueryTs and RefTs Same)

	Index	0	1	2	3	4	5	6
Index	Partition	[5,8]	[8,7]	[7,6]	[6,9]	[9,1]	[1,3]	[3,5]
0	[5,8]	**0.0**	**3.2**	**2.8**	**1.4**	**8.1**	**6.4**	**3.6**
1	[8,7]	**3.2**	**0.0**	**1.4**	**2.8**	**6.1**	**8.1**	**5.4**
2	[7,6]	**2.8**	**1.4**	**0.0**	**3.2**	**5.4**	**6.7**	**4.1**
3	[6,9]	**1.4**	**2.8**	**3.2**	**0.0**	**8.5**	**7.8**	**5.0**
4	[9,1]	**8.1**	**6.1**	**5.4**	**8.5**	**0.0**	**8.2**	**7.2**
5	[1,3]	**6.4**	**8.1**	**6.7**	**7.8**	**8.2**	**0.0**	**2.8**
6	[3,5]	**3.6**	**5.4**	**4.1**	**5.0**	**7.2**	**2.8**	**0.0**
Matrix Profile		**1.4**	**1.4**	**1.4**	**1.4**	**5.4**	**2.8**	**2.8**
Location array		**3**	**2**	**1**	**0**	**2**	**6**	**5**

11. Repeat it until having exact result.

Note: If it is still repeating calculated, the matrix profile and location array are still **approximated result that converge to the exact result**.

MASS Algorithm:

Algorithm to calculate distance function while sliding window.

Z-Normalized Euclidean Distance in MASS func.

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

- It is empirical assumption to z-normalization for more obvious result.
- The relation equation between correlation coefficient is same as Z-Normalized euclidean distance func.

STAMP Calculation method:

- It uses MASS Algorithm to calculate distance function.
- It randomly select ith query and be setted sample size. Thus, it is an anytime algorithm.

Anytime algorithm: can be stopped but still give result like machine learning.

STOMP Calculation method:

$$d_{i,j} = \sqrt{2m\left(1 - \frac{T_i T_j - m\mu_i \mu_j}{m\sigma_i \sigma_j}\right)} \qquad T_{i+1}T_{j+1} = T_i T_j - t_i t_j + t_{i+m} t_{j+m}$$

For STOMP method, it **no need** to **recalculate product of TiTj**. But the query is calculated orderly.

Algorithms and Calculation method different by distance profile function

```
def matrix_profile(self,QueryTs,RefTs,type):
  QueryTs,RefTs  = self._timeserie_preprocessing(QueryTs,RefTs)
  mp,idx = self._inf_mp_and_index_arr(QueryTs,RefTs)

  order_arr = self._order_array(RefTs)
  for i in order_arr:
    (distanceProfile,id) = distanceProfileFunc(tsA,idx,m,tsB)

    idsToUpdate = distanceProfile < mp
    idx[idsToUpdate] = id[idsToUpdate]

    mp = np.minimum(mp,distanceProfile)

  return mp,idx
```

Algorithm	Calculation Time
Naive	1.503s
STAMP	0.002s
STOMP	0.57s

Task for comparison: compute **matrix profile function** for **one time serie** input of length **100** timepoints.