Package ‘matrixprofiler’

October 14, 2026

Type Package

Title Matrix Profile for R

Version 0.1.12

Maintainer Francisco Bischoff <fbischoff@med.up.pt>

Description This is the core functions needed by the 'tsmp' package. The low level and carefully checked mathematical functions are here. These are implementations of the Matrix Profile concept that was created by CS-UCR <http://www.cs.ucr.edu/~eamonn/MatrixProfile.html>.

License GPL-3

URL https://matrix-profile-foundation.github.io/matrixprofiler/, https://github.com/matrix-profile-foundation/matrixprofiler

BugReports https://github.com/matrix-profile-foundation/matrixprofiler/issues

Depends R (>= 4.1)

Imports checkmate (>= 2.1.0), Rcpp (>= 1.0.9)

Suggests debugme (>= 1.1.0), spelling (>= 2.2.0), testthat (>= 3.1.4)

LinkingTo Rcpp (>= 1.0.9), RcppThread (>= 2.1.3)

Config/testthat/edition 3

Config/testthat/parallel false

Encoding UTF-8

Language en-US

LazyData true

NeedsCompilation yes

SystemRequirements GNU make

Config/roxygen2/version 8.1.0

Author Francisco Bischoff [aut, cre] (ORCID: <https://orcid.org/0000-0002-5301-8672>), Michael Yeh [res, ccp, ctb] (ORCID: <https://orcid.org/0000-0002-9807-2963>), Diego Silva [res, ccp, ctb] (ORCID:

<https://orcid.org/0000-0002-5184-9413>),
Yan Zhu [res, ccp, ctb] (ORCID:
<https://orcid.org/0000-0002-5952-2108>),
Hoang Dau [res, ccp, ctb] (ORCID:
<https://orcid.org/0000-0003-2439-5185>),
Michele Linardi [res, ccp, ctb] (ORCID:
<https://orcid.org/0000-0002-3249-2068>)

Repository CRAN

Date/Publication 2026-10-14 04:20:02 UTC

Contents

contrast 2
has_cuda 4
mass 4
matrixprofiler 6
motifs_discords_small 7
mov_mean 8
mpdist 11
mstomp 12
rfcp 13
rfcp_top_k 15
stamp 18
znorm 23

Index **26**

`contrast` *Contrast Profile Computation*

Description

Computes the contrast profile of two (classes of) time series.

Usage

```
contrast(
  negative_data,
  positive_data,
  window_size,
  positive_matrix = NULL,
  exclusion_zone = 0.5,
  distance = c("euclidean", "pearson"),
  n_workers = 1L,
  progress = TRUE
)
```

Arguments

negative_data	Required. Any 1-dimension series of numbers (matrix, vector, ts etc.) where the pattern is not present
positive_data	Required. Any 1-dimension series of numbers (matrix, vector, ts etc.) where the pattern is present
window_size	Required. An integer defining the rolling window size.
positive_matrix	Optional. A precomputed self-similar matrix profile of the positive data.
exclusion_zone	A numeric. Defines the size of the area around the rolling window that will be ignored to avoid trivial matches. Default is 0.5, i.e., half of the window_size.
distance	A string. Currently accepts euclidean and pearson. Defaults to euclidean.
n_workers	An integer. The number of threads using for computing. Defaults to 1. For parallel execution across processes, prefer a PSOCK cluster and use n_workers = 1 within each worker. See matrixprofiler for fork limitations.
progress	A logical. If TRUE (the default) will show a progress bar. Useful for long computations.

Details

Constrast Profile:

This algorithm returns the contrast profile of two time series, which shows the position of patters that are similar in the positive data, but at the same time very dissimilar in the negative data. In other words, this means that such a pattern represents well positive data and may be taken as a "signature" of that class. More information can be found in the references.

Value

Returns a list with the contrast_profile, plato, plato_nn, plato_idx, plato_nn_idx, w, ez, euclidean values

References

- R. Mercer, S. Alaee, A. Abdoli, S. Singh, A. Murillo and E. Keogh, "Matrix Profile XXIII: Contrast Profile: A Novel Time Series Primitive that Allows Real World Classification," 2021 IEEE International Conference on Data Mining (ICDM), 2021, pp. 1240-1245, doi: 10.1109/ICDM51629.2021.00151.

Website: http://www.cs.ucr.edu/~eamonn/MatrixProfile.html

Examples

```
cp <- contrast(motifs_discords_small, rev(motifs_discords_small), 50)
```

has_cuda — *Check if package was compiled with CUDA support*

Description

Queries whether `matrixprofiler` was compiled with NVIDIA CUDA GPU acceleration (`MATRIXPROFILER_ENABLE_CUDA=1`

Usage

```
has_cuda()
```

Details

By default, `matrixprofiler` is compiled without CUDA to maintain portability and compliance with CRAN policies, running all algorithms on multi-threaded CPU engines powered by `RcppThread`.

If an NVIDIA GPU and the CUDA Toolkit are available on the host system, the package can be recompiled with CUDA acceleration for supported algorithms (such as `rfcp` and `mpx`) using the helper script `inst/cuda/build_cuda_rcpp.sh` or by setting the environment variables:

- `MATRIXPROFILER_ENABLE_CUDA=1`
- `MATRIXPROFILER_CUDA_OBJECT` pointing to precompiled CUDA object files (`.o`)
- `MATRIXPROFILER_CUDA_LIBS` pointing to the CUDA runtime library (e.g. `-lcudart`)

When `has_cuda()` returns `FALSE`, attempting to execute an algorithm with `backend = "cuda"` will stop with an informative error instructing how to enable CUDA.

Value

A logical value: `TRUE` if CUDA support was enabled at compile time, `FALSE` otherwise.

Examples

```
has_cuda()
```

mass — *Computes the Distance between the 'data' and the 'query'.*

Description

This algorithm will use a rolling window, to computes the distance thorough the whole data. This means that the minimum distance found is the *motif* and the maximum distance is the *discord* on that time series. **Attention** you need first to create an object using `mass_pre()`. Read below.

Usage

```
mass(
  pre_obj,
  data,
  query = data,
  index = 1,
  version = c("v3", "v2"),
  n_workers = 1
)

mass_pre(
  data,
  window_size,
  query = NULL,
  type = c("normalized", "non_normalized", "absolute", "weighted"),
  weights = NULL
)
```

Arguments

`pre_obj`	Required. This is the object resulting from `mass_pre()`. The is no *MASS* without a *pre*.
`data`	Required. Any 1-dimension series of numbers (`matrix`, `vector`, `ts` etc.)
`query`	Optional. Accepts the same types as `data` and is used for join-similarity. Defaults to `data` for self-similarity. **IMPORTANT** Use the same data used on `mass_pre()`.
`index`	An `integer`. This is the index of the rolling window that will be used. Must be between `1` and `length(data) - window_size + 1`.
`version`	A `string`. Chooses the version of MASS what will be used. Ignored if `mass_pre()` is not the "normalized" type.
`n_workers`	An `integer`. The number of threads used for computing. Defaults to `1`. For parallel execution across processes, prefer a PSOCK cluster and use `n_workers = 1` within each worker. See matrixprofiler for fork limitations.
`window_size`	Required. An integer defining the rolling window size.
`type`	This changes how the MASS algorithm will compare the rolling window and the data. (See details).
`weights`	Optional. It is used when the `type` is `weighted`, and has to be the same size as the `window_size`.

Details

There are currently four ways to compare the window with the data:

1. **normalized**: this normalizes the data and the query window. This is the most frequently used.
2. **non_normalized**: this won't normalize the query window. The data still being normalized.
3. **absolute**: this won't normalize both the data and the query window.

4. **weighted**: this normalizes the data and query window, and also apply a weight vector on the query.

Value

`mass()` returns a `list` with the `distance_profile` and the `last_product` that is only useful for computing the Matrix Profile.

`mass_pre()` returns a `list` with several precomputations to be used on MASS later. **Attention** use this before `mass()`.

Examples

```
pre <- mass_pre(motifs_discords_small, 50)
dist_profile <- mass(pre, motifs_discords_small)
pre <- mass_pre(motifs_discords_small, 50)
dist_profile <- mass(pre, motifs_discords_small)
```

`matrixprofiler` *Matrix Profile for R*

Description

This package is derived from the former package `tsmp`. It is intended to make a clear separation of what is the Matrix Profile computation and what are the data mining process we can do using Matrix Profile.

Details

The Matrix Profile, has the potential to revolutionize time series data mining because of its generality, versatility, simplicity and scalability. In particular it has implications for time series motif discovery, time series joins, shapelet discovery (classification), density estimation, semantic segmentation, visualization, rule discovery, clustering etc.

Multithreading

This package uses RcppThread for multithreaded computations. Thread counts can be controlled directly via the `n_workers` argument in each function.

Parallel execution and forked processes

`matrixprofiler` uses native threads for parallel computation. To parallelize a single computation, use the function's `n_workers` argument.

Avoid calling package functions from fork-based workers, such as those created by `parallel::mclapply()`, particularly after multithreaded libraries have been initialized in the parent R session. Forked processes inherit library state that may cause deadlocks or crashes.

Where fork detection is implemented, requesting multiple workers in a forked child triggers a warning and reduces `n_workers` to `1`. This precaution does not make fork-based execution safe, because it cannot undo inherited library state.

To process multiple time series concurrently, prefer a PSOCK cluster, which starts independent R sessions (see `parallel::makePSOCKcluster()`). Use `n_workers = 1` within each worker to avoid excessive thread creation.

This limitation concerns the execution environment and applies to both STOMP and MPX; switching algorithms does not remove it.

References

- Yeh CCM, Zhu Y, Ulanova L, Begum N, Ding Y, Dau HA, et al. Matrix profile I: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets. Proc - IEEE Int Conf Data Mining, ICDM. 2017;1317-22.

- Zhu Y, Imamura m, Nikovski D, Keogh E. Matrix Profile VII: Time Series Chains: A New Primitive for Time Series Data Mining. Knowl Inf Syst. 2018 Jun 2;1-27.

- Zhu Y, Zimmerman Z, Senobari NS, Yeh CM, Funning G. Matrix Profile II : Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins. Icdm. 2016 Jan 22;54(1):739-48.

Website: http://www.cs.ucr.edu/~eamonn/MatrixProfile.html

`motifs_discords_small` *Just a synthetic dataset for testing*

Description

Just a synthetic dataset for testing

Usage

```
motifs_discords_small
```

Format

A `vector` with 875 observations

mov_mean *Several moving window functions*

Description

These functions compute running window statistics with comprehensive support for missing and non-finite values (NA, NaN, Inf).

Usage

```
mov_mean(
  data,
  window_size,
  type = c("ogita", "normal", "weighted", "fading"),
  eps = 0.9,
  na_strategy = c("window_na", "remove", "error"),
  na.rm = NULL
)

mov_var(
  data,
  window_size,
  type = c("ogita", "normal", "weighted", "fading"),
  eps = 0.9,
  na_strategy = c("window_na", "remove", "error"),
  na.rm = NULL
)

mov_sum(
  data,
  window_size,
  type = c("ogita", "normal", "weighted", "fading"),
  eps = 0.9,
  na_strategy = c("window_na", "remove", "error"),
  na.rm = NULL
)

mov_max(
  data,
  window_size,
  na_strategy = c("window_na", "remove", "error"),
  na.rm = NULL
)

mov_min(
  data,
  window_size,
```

```
  na_strategy = c("window_na", "remove", "error"),
  na.rm = NULL
)

mov_std(
  data,
  window_size,
  rcpp = TRUE,
  na_strategy = c("window_na", "remove", "error"),
  na.rm = NULL
)

movmean_std(
  data,
  window_size,
  rcpp = TRUE,
  na_strategy = c("window_na", "remove", "error"),
  na.rm = NULL
)

muinvn(
  data,
  window_size,
  n_workers = 1L,
  na_strategy = c("window_na", "remove", "error"),
  na.rm = NULL
)

zero_crossing(data, window_size)
```

Arguments

`data` A `vector` or a column `matrix` of `numeric`.

`window_size` An `integer`. The size of the rolling window.

`type` A `string`. Select between several algorithms. Default is `ogita` (See details).

`eps` A `numeric`. Used only for fading algorithms (See details), otherwise has no effect.

`na_strategy` A `string` indicating how missing values (`NA`/`NaN`) are handled:

- `"window_na"` (default): Any window touching an `NA`/`NaN` yields `NA_real_`, without contaminating adjacent complete windows or stopping with an error.
- `"remove"`: Missing values inside the window are ignored, computing the statistic over remaining finite values.
- `"error"`: Throws an error immediately if missing values are present.

`na.rm` A `logical`. Standard R compatibility shortcut. If provided, `TRUE` maps to `na_strategy = "remove"`, and `FALSE` maps to `na_strategy = "window_na"`.

rcpp	A `logical`. If `TRUE` will use the Rcpp implementation, otherwise will use the R implementation, that may or not be slower.
n_workers	An `integer`. The number of threads using for computing. Defaults to `1`. For parallel execution across processes, prefer a PSOCK cluster and use `n_workers = 1` within each worker. See matrixprofiler for fork limitations.

Details

Some functions may use different algorithms to compute the results. The available types are:

1. **ogita**: This is the default. It uses the Ogita *et al.*, Accurate Sum, and Dot Product for precision. It is not the fastest algorithm, but the time spent vs. guarantee of precision worth it.
2. **normal**: This uses the `cumsum` method that is faster, but unreliable in some situations (I have to find the references, but is true).
3. **weighted**: This uses Rodrigues P., *et al.* algorithm that uses a weighted window for online purposes. The `eps` argument controls the factor. (The function is not online yet)
4. **fading**: This also uses Rodrigues P., *et al.* algorithm that in this case, uses a fading factor, also for online purposes. he `eps` argument controls the factor. (The function is not online yet)

Another important detail is that the *standard deviation* we use for all computations is the *population* (i.e.: divided by `n`), not the *sample* (i.e.: divided by `n - 1`). That is why we also provide the internally the `:::std()` function that computes the *population*, differently from `stats::sd()` that is the *sample* kind. Further more, `movmean_std()` shall be used when you need both results in one computation. This is faster than call `mov_mean()` followed by `mov_std()`. Finally, `muinvn()` is kept like that for historical reasons, as it is the function used by `mpx()`. It returns the `sig` (stable inverse centered norm) instead of `std` (`sig` is equals to `1 / (std * sqrt(window_size))`).

Value

`mov_mean()` returns a `vector` with moving `avg`.

`mov_var()` returns a `vector` with moving `var`.

`mov_sum()` returns a `vector` with moving `sum`.

`mov_max()` returns a `vector` with moving `max`.

`mov_min()` returns a `vector` with moving `min`.

`mov_std()` returns a `vector` with moving `sd`.

`movmean_std()` returns a `list` with `vectors` of the moving `avg`, `sd`, `sig`, `sum` and `sqrsum`.

`muinvn()` returns a `list` with `vectors` of moving `avg` and `sig`.

`zero_crossing()` returns a `vector` of times the data crossed the 'zero' line inside a rolling window.

Examples

```
mov <- mov_mean(motifs_discords_small, 50)
mov <- mov_var(motifs_discords_small, 50)
mov <- mov_sum(motifs_discords_small, 50)
mov <- mov_max(motifs_discords_small, 50)
mov <- mov_min(motifs_discords_small, 50)
```

```
mov <- mov_std(motifs_discords_small, 50)
mov <- movmean_std(motifs_discords_small, 50)
mov <- muinvn(motifs_discords_small, 50)
zero_cross <- zero_crossing(motifs_discords_small, 50)
```

---

mpdist *MPdist - Distance between Time Series using Matrix Profile*

Description

MPdist is a distance measure which considers two time series to be similar if they share many similar subsequences, regardless of the order of matching subsequences.

Usage

```
mpdist(
  ref_data,
  query_data,
  window_size,
  type = c("simple", "vector"),
  thr = 0.05,
  n_workers = 1L
)
```

Arguments

`ref_data`	a `matrix` or a `vector`. The reference data.
`query_data`	a `matrix` or a `vector`. The query data.
`window_size`	an `int`. Size of the sliding window. Must be at least 4.
`type`	the type of result: `"simple"` (default) returns a scalar distance, `"vector"` returns distance between all sliding windows.
`thr`	threshold for MPdist. (Default is `0.05`).
`n_workers`	an `int`. Number of threads for parallel computation. (Default is `1`). For parallel execution across processes, prefer a PSOCK cluster and use `n_workers = 1` within each worker. See matrixprofiler for fork limitations.

Value

Returns the distance scalar (`"simple"`) or a numeric vector (`"vector"`).

References

- Gharghabi S, Imani S, Bagnall A, Darvishzadeh A, Keogh E. Matrix Profile XII: MPdist: A Novel Time Series Distance Measure to Allow Data Mining in More Challenging Scenarios. In: 2018 IEEE International Conference on Data Mining (ICDM). 2018.

Website: https://sites.google.com/site/mpdistinfo/

Examples

```
ref_data <- motifs_discords_small[1:200]
qe_data <- motifs_discords_small[201:400]
w <- 30
deq <- mpdist(ref_data, qe_data, w)
```

---

mstomp *Multivariate STOMP Algorithm*

Description

Computes the Matrix Profile and Profile Index for Multivariate Time Series.

Usage

```
mstomp(
  data,
  window_size,
  exclusion_zone = 0.5,
  must_dim = NULL,
  exc_dim = NULL,
  n_workers = 1L,
  progress = TRUE
)
```

Arguments

data	a matrix of numeric, where each column is a time series. Accepts vector, list and data.frame too.
window_size	an int with the size of the sliding window.
exclusion_zone	a numeric. Size of the exclusion zone, based on window size (default is 0.5).
must_dim	an int or vector of which dimensions to forcibly include (default is NULL).
exc_dim	an int or vector of which dimensions to exclude (default is NULL).
n_workers	an int. Number of workers for parallel execution (default is 1). For parallel execution across processes, prefer a PSOCK cluster and use n_workers = 1 within each worker. See matrixprofiler for fork limitations.
progress	a logical. If TRUE will show a progress bar. (Default is TRUE).

Value

Returns a list with the matrix profile matrix_profile, profile index profile_index, left and right matrix profiles left_matrix_profile, right_matrix_profile and profile indices left_profile_index, right_profile_index, window size w, number of dimensions n_dim, exclusion zone ez, must dimensions must and excluded dimensions exc.

References

- Yeh CM, Kavantzas N, Keogh E. Matrix Profile VI : Meaningful Multidimensional Motif Discovery.

- Zhu Y, Imamura M, Nikovski D, Keogh E. Matrix Profile VII: Time Series Chains: A New Primitive for Time Series Data Mining. Knowl Inf Syst. 2018 Jun 2;1-27.

Website: https://sites.google.com/view/mstamp/

Website: http://www.cs.ucr.edu/~eamonn/MatrixProfile.html

See Also

Other matrix profile computations: stamp()

Examples

```
mp <- mstomp(motifs_discords_small, 50, progress = FALSE)
```

---

rfcp *Relative Frequency Contrast Profile*

Description

Computes an exact, NA-aware Relative Frequency Contrast Profile (RFCP) between a positive series and a negative series. For every valid positive window, RFCP compares ordered non-overlapping neighbor ranks from the positive self-join and the positive-to-negative AB-join.

Usage

```
rfcp(
  positive_data,
  negative_data,
  window_size,
  max_freq,
  exclusion_radius = NULL,
  query_begin = 1L,
  query_end = NULL,
  return_profiles = FALSE,
  n_workers = 1L,
  progress = TRUE,
  backend = "cpu"
)
```

Arguments

- `positive_data` Numeric positive/query time series.
- `negative_data` Numeric negative/reference time series.
- `window_size` Integer subsequence length, at least `2` and no larger than either input series.
- `max_freq` Integer number of non-overlapping neighbor ranks requested.
- `exclusion_radius` Integer exclusion radius between accepted neighbor starts. `NULL` uses `window_size`, the RFCP protocol setting.
- `query_begin` First positive query start to process, using one-based inclusive coordinates. Defaults to the first start.
- `query_end` Last positive query start to process, using one-based inclusive coordinates. `NULL` processes through the final start.
- `return_profiles` Whether to return the complete RFMP, RFCP, and index matrices for the requested query block. This validation mode is limited to 5,000,000 rank-query cells. Defaults to `FALSE`.
- `n_workers` Number of native worker threads for the CPU backend. Ignored when `backend = "cuda"`. Defaults to `1`. For parallel execution across processes, prefer a PSOCK cluster and use `n_workers = 1` within each worker. See matrixprofiler for fork limitations.
- `progress` Logical progress flag. Reserved for native progress reporting; defaults to `TRUE`.
- `backend` Computation backend: `"cpu"` (default) or `"cuda"`. The CUDA backend is optional and requires a local build with CUDA enabled (see `inst/cuda/build_cuda_rcpp.sh`).

Details

`NA`, `NaN`, `Inf`, and `-Inf` are barriers. Windows touching a barrier, constant windows, and otherwise non-normalizable windows cannot be queries or neighbors. Returned neighbor indices are one-based global coordinates. Candidates are ordered by increasing distance and then increasing index. After accepting a neighbor at `j`, the closed interval `[j - exclusion_radius, j + exclusion_radius]` is excluded. The same neighborhood around the query is excluded before AA selection.

For rank `k`, distances are clipped at `sqrt(2 * window_size)` before calculating `max((AB - AA) / sqrt(2 * window_size), 0)`. Missing ranks contribute zero while the RMSC denominator remains the requested `max_freq`. `effective_max_freq` is diagnostic: it is the largest number of paired finite AA/AB ranks observed for any completed query and never changes the denominator.

The default mode materializes only the RMSC values for the requested query block and the rank vectors for the selected Plato. Blocks use global query coordinates and can be concatenated for checkpointed execution. With `return_profiles = TRUE`, RFMP matrices contain unclipped z-normalized Euclidean distances and RFCP contains the corresponding clipped contrasts. Native query tiles use `min(32768, max(1024, 16 * window_size))`, further capped at the ceiling of the requested query count divided by the effective worker count. The first tile starts at `query_begin`, so checkpoint blocks do not perform discarded warm-up rows. The CUDA backend executes the full join query-by-query with optional profile matrices returned identically to the CPU backend.

Value

A list with RFCP outputs:

- `rms_profile`: numeric vector of root-mean-square contrasts.
- `valid_window_positive`: logical mask of valid query windows.
- `valid_window_negative`: logical mask of valid reference windows.
- `plato_index`: one-based index of the highest-contrast query.
- `plato_rms`: contrast score at `plato_index`.
- `plato_twin_index`: one-based index of the nearest self-join neighbor at `plato_index`.
- `rfcp_at_plato`: clipped contrasts across accepted neighbor ranks.
- `rfmp_aa_at_plato`: positive self-join distances at plato.
- `rfmp_ab_at_plato`: negative AB distances at plato.
- `rfmpi_aa_at_plato`: positive neighbor indices at plato.
- `rfmpi_ab_at_plato`: negative neighbor indices at plato.
- `requested_max_freq`: requested rank count.
- `effective_max_freq`: highest paired rank observed.
- `query_begin`, `query_end`: one-based query range.
- `completed_query_end`: end of completed queries.
- `partial`: whether computation stopped early.
- `rfmp_aa`, `rfmp_ab`, `rfcp`, `rfmpi_aa`, `rfmpi_ab`: optional full profile matrices when `return_profiles = TRUE`.

Examples

```
pos <- c(sin(seq(0, 4 * pi, length.out = 100)))
neg <- c(cos(seq(0, 10 * pi, length.out = 150)))
res <- rfcp(pos, neg, window_size = 10, max_freq = 2)
res$plato_index
```

`rfcp_top_k` *Top-K Relative Frequency Contrast Profile*

Description

Discovers the top-K contrasting patterns (Platos) between a positive time series and a negative time series using an iterative Relative Frequency Contrast Profile (RFCP) procedure.

Usage

```
rfcp_top_k(
  positive_data,
  negative_data,
  window_size,
  k_platos = 2L,
  max_freq = 5L,
  exclusion_radius = NULL,
  min_rms = 1e-04,
  adaptive_max_freq = FALSE,
  min_freq = 2L,
  n_workers = 1L,
  backend = "cpu",
  progress = FALSE,
  query_block_size = NULL,
  progress_callback = NULL,
  checkpoint_callback = NULL,
  resume_state = NULL
)
```

Arguments

`positive_data` Numeric positive/query time series.

`negative_data` Numeric negative/reference time series.

`window_size` Integer subsequence length, at least 2 and no larger than either input series.

`k_platos` Integer number of distinct Platos to discover. Defaults to `2L`.

`max_freq` Integer scalar or integer vector specifying the number of non-overlapping neighbor ranks requested for each Plato iteration. If a scalar is provided, it is used for all iterations. If a vector is provided, its length must equal `k_platos`, allowing dynamic re-evaluation of frequency for rarer morphologies. Defaults to `5L`.

`exclusion_radius` Integer exclusion radius between accepted neighbor starts and for padding concatenated segments. `NULL` uses `window_size`.

`min_rms` Numeric threshold for root-mean-square contrast. Iteration stops early if the detected Plato has an RMS contrast below this value. Defaults to `1e-4`.

`adaptive_max_freq` Logical. If `TRUE`, dynamically adapts `max_freq` for subsequent iterations based on the observed support (number of valid positive self-join neighbors with distance below $\sqrt{2 \cdot \text{window_size}}$) of the current Plato. Defaults to `FALSE`.

`min_freq` Integer minimum rank frequency floor when `adaptive_max_freq = TRUE`. Defaults to `2L`.

`n_workers` Number of native worker threads for the CPU backend. Ignored when `backend = "cuda"`. Defaults to `1L`. For parallel execution across processes, prefer a PSOCK cluster and use `n_workers = 1` within each worker. See matrixprofiler for fork limitations.

`backend` Computation backend: `"cpu"` (default) or `"cuda"`.

`progress` Logical progress flag for each RFCP iteration. Defaults to `FALSE`.

`query_block_size` Optional integer query block size for chunked execution. If `NULL` (default), queries are processed monolithically in a single call. If specified, queries are processed in contiguous blocks of size `query_block_size`.

`progress_callback` Optional callback function `function(event)` invoked periodically with structured progress events (`"round_start"`, `"block_done"`, `"round_done"`).

`checkpoint_callback` Optional callback function `function(state)` invoked after each block and Plato with a serializable state list.

`resume_state` Optional state list from a previous execution (`checkpoint_callback`) used to resume execution idempotently.

Details

In each iteration $i \in \{1, \ldots, K\}$, the algorithm:

1. Executes `rfcp` using the current augmented negative series and the requested frequency `max_freq[i]`.
2. Identifies the highest-contrast query window (Plato$_i$) outside the exclusion zone of previously discovered Platos and their associated neighbors.
3. If the contrast score is below `min_rms` or no valid candidate remains, iteration terminates early.
4. Extracts the subsequences of Plato$_i$ and all its accepted positive self-join neighbors (`rfmpi_aa_at_plato`), expanded by the exclusion zone:

$$[\max(1, j - \text{exclusion_radius}), \min(N^{(+)}, j + \text{window_size} - 1 + \text{exclusion_radius})]$$

5. Merges overlapping intervals to maintain signal continuity, and appends the resulting segments to the negative series separated by `NA` barriers. Because `rfcp` is NA-aware, this suppresses the contrast of that morphology and its trivial overlaps in subsequent iterations without introducing spurious cross-boundary windows.
6. If `adaptive_max_freq = TRUE`, re-evaluates the frequency for the next iteration based on the count of finite AA neighbors with distance below $\sqrt{2 \cdot \text{window_size}}$, bounded below by `min_freq`.

Value

An object of class `"rfcp_top_k"`, which is a list containing:

- `k_requested`: requested number of Platos.
- `k_found`: number of Platos discovered before stopping.
- `window_size`: subsequence length.
- `exclusion_radius`: exclusion radius used.
- `adaptive_max_freq`: whether adaptive frequency was enabled.
- `min_freq`: minimum frequency floor used.
- `max_freq`: vector of frequencies used for each Plato.

- `platos`: list of discovered Plato objects, each containing:
  - `k`: Plato iteration rank (1, 2, ...).
  - `plato_index`: one-based start index in `positive_data`.
  - `plato_rms`: contrast score at `plato_index`.
  - `plato_subsequence`: numeric vector containing the pattern.
  - `plato_twin_index`: one-based start index of nearest self-join neighbor.
  - `rfcp_at_plato`: clipped contrasts across accepted ranks.
  - `rfmp_aa_at_plato`: positive self-join distances.
  - `rfmp_ab_at_plato`: negative AB distances.
  - `rfmpi_aa_at_plato`: positive neighbor indices.
  - `rfmpi_ab_at_plato`: negative neighbor indices.
  - `effective_max_freq`: highest paired rank observed.
- `negative_data_final`: the negative series augmented with the suppressed Plato occurrences and exclusion zones.

Examples

```
set.seed(42)
# Create positive series with 5 sine pulses and 2 cosine pulses
t <- seq(0, 4 * pi, length.out = 20)
sine_pulse <- sin(t)
cos_pulse <- cos(t)
pos <- rnorm(300, sd = 0.05)
for (p in c(20, 60, 100, 140, 180)) pos[p:(p + 19)] <- pos[p:(p + 19)] + sine_pulse
for (p in c(220, 260)) pos[p:(p + 19)] <- pos[p:(p + 19)] + cos_pulse * 1.5
neg <- rnorm(300, sd = 0.05)

top_platos <- rfcp_top_k(pos, neg, window_size = 20, k_platos = 2, max_freq = c(4, 2))
print(top_platos)
```

`stamp` *Matrix Profile Computation*

Description

STAMP Computes the best so far Matrix Profile and Profile Index for Univariate Time Series.

STOMP is a faster implementation with the caveat that is not anytime as STAMP or SCRIMP.

SCRIMP is a faster implementation, like STOMP, but has the ability to return anytime results as STAMP.

MPX is by far the fastest implementation with the caveat that is not anytime as STAMP or SCRIMP.

Usage

```
stamp(
  data,
  window_size,
  query = NULL,
  exclusion_zone = 0.5,
  s_size = 1,
  n_workers = 1,
  progress = TRUE
)

stomp(
  data,
  window_size,
  query = NULL,
  exclusion_zone = 0.5,
  n_workers = 1,
  progress = TRUE,
  left_right_profile = FALSE
)

scrimp(
  data,
  window_size,
  query = NULL,
  exclusion_zone = 0.5,
  s_size = 1,
  pre_scrimp = 0.25,
  n_workers = 1,
  progress = TRUE
)

mpx(
  data,
  window_size,
  query = NULL,
  exclusion_zone = 0.5,
  s_size = 1,
  idxs = TRUE,
  distance = c("euclidean", "pearson"),
  n_workers = 1,
  progress = TRUE,
  na_strategy = c("monolithic", "segmented", "segmented_native"),
  backend = c("cpu", "cuda")
)

mpx_stream_right(
  new_data,
```

```
  start_obj,
  history = 0,
  mp_time_constraint = 0,
  progress = TRUE
)

mpx_stream_s_right(
  data,
  batch_size,
  start_obj,
  stats,
  history = 0,
  mp_time_constraint = 0,
  progress = TRUE,
  threshold = -1
)
```

Arguments

data	Required. Any 1-dimension series of numbers (matrix, vector, ts etc.) (See details).
window_size	Required. An integer defining the rolling window size.
query	Optional. Another 1-dimension series of numbers for an AB-join similarity. Default is NULL (See details).
exclusion_zone	A numeric. Defines the size of the area around the rolling window that will be ignored to avoid trivial matches. Default is 0.5, i.e., half of the window_size.
s_size	A numeric. Used on anytime algorithms (stamp, scrimp, mpx) if only part of the computation is needed. Default is 1.0 (means 100%). For MPX specifically, values below 1 use a randomized subset of diagonals and are the recommended setting for an anytime approximation. At 1, MPX computes every diagonal in monotone order for performance; the completed result is exact, but an interrupted partial result can be less representative than a randomized partial run.
n_workers	An integer. The number of threads using for computing. Defaults to 1. For parallel execution across processes, prefer a PSOCK cluster and use n_workers = 1 within each worker. See matrixprofiler for fork limitations.
progress	A logical. If TRUE (the default) will show a progress bar. Useful for long computations. (See details)
left_right_profile	(stomp() only) A boolean. If TRUE, the function will return the left and right profiles.
pre_scrimp	A numeric. If not zero, pre_scrimp is computed, using a fraction of the data. Default is 0.25. This parameter is ignored when using multithread or AB-join.
idxs	(mpx() only) A logical. Specifies if the computation will return the Profile Index or not. Defaults to TRUE.
distance	(mpx() only) A string. Currently accepts euclidean and pearson. Defaults to euclidean.

`na_strategy` (`mpx()` only) Strategy for a parallel NA-aware join: `"monolithic"` (the default), `"segmented"`, or `"segmented_native"`. The native strategy supports sampled diagonal tiles when `s_size < 1`, marks the result as partial, and shares one native pre-calculation and scheduler across finite blocks. The native strategy also masks constant or otherwise non-normalizable windows. The `"segmented"` reference strategy remains restricted to inputs whose invalid windows are caused only by non-finite barriers.

`backend` (`mpx()` only) Computation backend: `"cpu"` (default) or `"cuda"`. The CUDA backend is optional, supports exact joins only (`s_size = 1`), and requires a local build with CUDA enabled.

`new_data` a vector with the new data

`start_obj` obj from another run

`history` an `int`. Max distance where to look for the best match in matrix profile. (default is NULL).

`mp_time_constraint` time constraint for the mp update in terms of indices. This parameter specifies how far back in the time series the algorithm will search for the nearest neighbor. A value of 0 (default) means no constraint and the algorithm will check the entire time series. Higher values limit the search to a window that extends `mp_time_constraint` indices into the past.

`batch_size` size of the new data

`stats` obj with stats for mpx

`threshold` Optional numeric threshold for distance comparison. Defaults to -1.0.

Details

The Matrix Profile, has the potential to revolutionize time series data mining because of its generality, versatility, simplicity and scalability. In particular it has implications for time series motif discovery, time series joins, shapelet discovery (classification), density estimation, semantic segmentation, visualization, rule discovery, clustering etc.

`progress`, it is really recommended to use it as feedback for long computations. It indeed adds some (neglectable) overhead, but the benefit of knowing that your computer is still computing is much bigger than the seconds you may lose in the final benchmark. Native multithreading uses RcppThread and is controlled by `n_workers`. See matrixprofiler for guidance on parallel execution across processes and fork limitations. The `data` and `query` parameters will be internally converted to a single vector using `as.numeric()`, thus, bear in mind that a multidimensional matrix may not work as you expect, but most 1-dimensional data types will work normally. If `query` is provided, expect the same pre-procesment done for `data`; in addition, `exclusion_zone` will be ignored and set to `0`. Both `data` and `query` doesn't need to have the same size and they can be interchanged if both are provided. The difference will be in the returning object. AB-Join returns the Matrix Profile 'A' and 'B' i.e., the distance between a rolling window from query to data and from data to query.

stamp:

The anytime STAMP computes the Matrix Profile and Profile Index in such manner that it can be stopped before its complete calculation and return the best so far results allowing ultra-fast approximate solutions.

stomp:
The STOMP uses a faster implementation to compute the Matrix Profile and Profile Index. It can be stopped earlier by the user, but the result is not considered anytime, just incomplete. For a anytime algorithm, use `stamp()` or `scrimp()`.

scrimp:
The SCRIMP algorithm was the anytime solution for stomp. It is as fast as stomp but allows the user to cancel the computation and get an approximation of the final result. The exact SCRIMP phase shares the NA-aware diagonal recurrence used by MPX. In serial mode, pre-scrimp first seeds the profile from sampled distance profiles; the multithreaded implementation skips this pre-scrimp stage.

mpx:
This algorithm was developed apart from the main Matrix Profile branch that relies on Fast Fourier Transform (FFT) at least in one part of the process. This algorithm doesn't use FFT at all and is several times faster. It also relies on Ogita's work for better precision computing mean and standard deviation (part of the process).

Inputs containing `NA`, `NaN`, `Inf`, or `-Inf` are automatically routed to the missing-data implementation. Fully finite inputs continue to use the faster regular MPX implementation.

For MPX, values of `s_size` below `1` select a randomized subset of diagonals and provide the intended anytime approximation. With `s_size = 1`, all diagonals are processed in monotone order for better locality, so the final result is exact but an interruption before completion can be less representative than a randomized partial run. Use `s_size < 1` when an interrupted result should remain an anytime approximation.

The native segmented parallel AA implementation has an optional cache-tuning setting for large NA-aware self-joins. This setting applies only when `na_strategy = "segmented_native"` selects the native NA-aware parallel path. When `s_size = 1`, `MATRIXPROFILER_NATIVE_AA_BLOCK_SIZE` controls the number of profile windows in each rectangular internal block. Its default is `8192`; values from `1024` to `262144` are accepted and invalid values fall back to the default. For example, set it for one shell invocation with `MATRIXPROFILER_NATIVE_AA_BLOCK_SIZE=16384 Rscript script.R`, or set it in R with `Sys.setenv(MATRIXPROFILER_NATIVE_AA_BLOCK_SIZE = "16384")` before calling `mpx()`. Remove the override with `Sys.unsetenv("MATRIXPROFILER_NATIVE_AA_BLOCK_SIZE")`. The setting is read at call time, does not change the public `mpx()` API or its numerical parameters, and is ignored for AB joins, finite inputs using regular MPX, and sampled native runs with `s_size < 1`. It primarily trades cache locality against task overhead; `8192` is a suitable starting point, while larger values should only be selected after benchmarking a substantially larger workload.

The internal C++ target `matrix_profile_native_pairs_per_task` (currently `100000000` pairwise window comparisons) controls the granularity of native parallel AB tasks and sampled AA tasks. It is used only by the native NA-aware parallel implementation. It is not an `mpx()` argument or an environment variable; changing it requires a source change and recompiling the package. It is normally left unchanged: increase it only when profiling a very large workload shows excessive scheduling overhead, while retaining several tasks per worker for load balancing. Neither setting affects the regular MPX paths or the serial NA-aware implementations.

Value

Returns a `list` with the `matrix_profile`, `profile_index` (if `idxs` is `TRUE` in `mpx()`), and some information about the settings used to build it, like `ez` and `partial` when the algorithm is finished early.

This document

Last updated on 2026-10-14 using R version 4.6.1.

References

- Yeh CCM, Zhu Y, Ulanova L, Begum N, Ding Y, Dau HA, et al. Matrix profile I: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets. Proc - IEEE Int Conf Data Mining, ICDM. 2017;1317-22.
- Zhu Y, Imamura m, Nikovski D, Keogh E. Matrix Profile VII: Time Series Chains: A New Primitive for Time Series Data Mining. Knowl Inf Syst. 2018 Jun 2;1-27.
- Zhu Y, Zimmerman Z, Senobari NS, Yeh CM, Funning G. Matrix Profile II : Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins. Icdm. 2016 Jan 22;54(1):739-48.

Website: http://www.cs.ucr.edu/~eamonn/MatrixProfile.html

See Also

`mass()` for the underlying algorithm that finds best match of a query.

`mpxab()` for the forward and reverse join-similarity.

Other matrix profile computations: `mstomp()`

Examples

```
mp <- stamp(motifs_discords_small, 50)
mp <- stomp(motifs_discords_small, 50)
mp <- scrimp(motifs_discords_small, 50)
mp <- mpx(motifs_discords_small, 50)
```

---

znorm *Math Functions*

Description

`znorm()`: Normalizes data for mean Zero and Standard Deviation One

`ed_corr()`: Converts euclidean distances into correlation values

`corr_ed()`: Converts correlation values into euclidean distances

`mode()`: Returns the most common value from a vector of integers

`std()`: Population SD, as R always calculate with n-1 (sample), here we fix it.

`normalize()`: Normalizes data to be between min and max.

`complexity()`: Computes the complexity index of the data

`binary_split()`: Creates a vector with the indexes of binary split.

Usage

```
znorm(data, rcpp = TRUE)

ed_corr(data, w, rcpp = TRUE)

corr_ed(data, w, rcpp = TRUE)

mode(x, rcpp = FALSE)

std(data, na.rm = FALSE, rcpp = TRUE)

normalize(data, min_lim = 0, max_lim = 1, rcpp = FALSE)

complexity(data)

binary_split(n, rcpp = TRUE)
```

Arguments

data	a vector of numeric.
rcpp	A logical. If TRUE will use the Rcpp implementation, otherwise will use the R implementation, that may or not be slower.
w	the window size
x	a vector of integers.
na.rm	A logical. If TRUE remove the NA values from the computation.
min_lim	A number
max_lim	A number
n	size of the vector

Value

znorm(): Returns the normalized data

ed_corr(): Returns the converted values from euclidean distance to correlation values.

corr_ed(): Returns the converted values from euclidean distance to correlation values.

mode(): Returns the most common value from a vector of integers.

std(): Returns the corrected standard deviation from sample to population.

normalize(): Returns the normalized data between min and max.

complexity(): Returns the complexity index of the data provided (normally a subset).

complexity(): Returns a vector with the binary split indexes.

Examples

```
normalized <- znorm(motifs_discords_small)
fake_data <- c(rep(3, 100), rep(2, 100), rep(1, 100))
correlation <- ed_corr(fake_data, 50)
fake_data <- c(rep(0.5, 100), rep(1, 100), rep(0.1, 100))
euclidean <- corr_ed(fake_data, 50)
fake_data <- c(1, 1, 4, 5, 2, 3, 1, 7, 9, 4, 5, 2, 3)
mode <- mode(fake_data)
fake_data <- c(1, 1.4, 4.3, 5.1, 2, 3.6, 1.24, 2, 9, 4.3, 5, 2.1, 3)
res <- std(fake_data)
fake_data <- c(1, 1.4, 4.3, 5.1, 2, 3.6, 1.24, 1, 9, 4.3, 5, 2.1, 3)
res <- normalize(fake_data)
fake_data <- c(1, 1.4, 4.3, 5.1, 2, 3.6, 1.24, 8, 9, 4.3, 5, 2.1, 3)
res <- complexity(fake_data)
fake_data <- c(10)
res <- binary_split(fake_data)
```

Index

∗ **datasets**
motifs_discords_small, 7
∗ **distance measure**
mpdist, 11
∗ **matrix profile computations**
mstomp, 12
stamp, 18

binary_split *(*znorm*)*, 23

complexity *(*znorm*)*, 23
contrast, 2
corr_ed *(*znorm*)*, 23

ed_corr *(*znorm*)*, 23

has_cuda, 4

mass, 4
mass_pre *(*mass*)*, 4
matrixprofiler, *3*, *5*, 6, *10–12*, *14*, *16*, *20*, *21*
mode *(*znorm*)*, 23
motifs_discords_small, 7
mov_max *(*mov_mean*)*, 8
mov_mean, 8
mov_min *(*mov_mean*)*, 8
mov_std *(*mov_mean*)*, 8
mov_sum *(*mov_mean*)*, 8
mov_var *(*mov_mean*)*, 8
movmean_std *(*mov_mean*)*, 8
mpdist, 11
mpx, *4*
mpx *(*stamp*)*, 18
mpx_stream_right *(*stamp*)*, 18
mpx_stream_s_right *(*stamp*)*, 18
mstomp, 12
mstomp(), *23*
muinvn *(*mov_mean*)*, 8

normalize *(*znorm*)*, 23

parallel::makePSOCKcluster(), *7*
parallel::mclapply(), *6*

rfcp, *4*, 13, *17*
rfcp_top_k, 15

scrimp *(*stamp*)*, 18
stamp, 18
stamp(), *13*
std *(*znorm*)*, 23
stomp *(*stamp*)*, 18

zero_crossing *(*mov_mean*)*, 8
znorm, 23