Package ‘statcomp’

October 29, 2019

**Title** Statistical Complexity and Information Measures for Time Series Analysis

**Version** 0.1.0

**Description** An implementation of local and global statistical complexity measures (aka Information Theory Quantifiers, ITQ) for time series analysis based on ordinal statistics (Bandt and Pompe (2002) <DOI:10.1103/PhysRevLett.88.174102>). Several distance measures that operate on ordinal pattern distributions, auxiliary functions for ordinal pattern analysis, and generating functions for stochastic and deterministic-chaotic processes for ITQ testing are provided.

**Depends** R (>= 2.7.0)

**License** GPL-2

**LazyData** true

**Imports** stats, zoo, Matrix, graphics

**RoxygenNote** 6.1.1

**NeedsCompilation** yes

**Author** Sebastian Sippel [aut, cre] (Original package development was supported by MPI Biogeochemistry, BGI Department), Holger Lange [aut], Fabian Gans [aut]

**Maintainer** Sebastian Sippel <sebastian.sippel@env.ethz.ch>

**Repository** CRAN

**Date/Publication** 2019-10-28 23:40:02 UTC

**R topics documented:**

Abbe .......................................................... 2
adjust_pattern ................................................ 3
complexity_Renyi .............................................. 4
fis ............................................................... 5
generate_lehmerperm_matrix ............................... 6
global_complexity .......................................... 6
hellinger_distance ........................................... 8
A function to compute Abbe values

Abbe

Description
Calculates "Abbe" values.

Usage
Abbe(x)

Arguments
x A time series
**Details**


**Value**

Abbe value

**Author(s)**

Sebastian Sippel

**References**

Tarnopolski et al. (2016), Physica A 461, 662-673.

**Examples**

```r
x = arima.sim(model=list(ar = 0.3), n = 10^4)
Abbe(x)
```

---

**adjust_pattern**

A function to create new pattern-coding schemes for the Fisher Information.

**Description**

Adjusts and reorders a pattern ordering matrix.

**Usage**

```r
adjust_pattern(pattern_matrix, adjustment)
```

**Arguments**

- `pattern_matrix`: A numeric matrix that specifies the pattern to be transformed into the position vector. ATTENTION: Pattern should be in the ranks permutation notation, otherwise does not really make sense.
- `adjustment`: A character vector, either adjustment = "jumps" or adjustment = "bitflips" that denotes the sorting type

**Details**

This function reorders permutations based on "jumps" or based on "bitflips".

**Value**

A numeric matrix that contains the permutation matrix.
complexity_Renyi

Author(s)

Sebastian Sippel

References


complexity_Renyi  A function to compute Renyi complexity

Description

Renyi complexity

Usage

complexity_Renyi(opd, alpha)

Arguments

opd  A numeric vector that details an ordinal pattern distribution.
alpha  alpha parameter in Renyi complexity

Details

This function calculates the Renyi complexity as described in Jauregui et al., Physica A, 498 74-85, 2018.

Value

The Renyi complexity value.

Author(s)

Sebastian Sippel

References


Examples

x = arima.sim(model=list(ar = 0.3), n = 10^4)
opd = ordinal_pattern_distribution(x = x, ndemb = 6)
complexity_Renyi(opd = opd, alpha = 0.5)
fis

A (low-level) function to compute the Fisher-information

Description

The function computes the Fisher information, i.e. a local information measure based on two different discretizations.

Usage

fis(opd, discretization)

Arguments

opd A numeric vector that details an ordinal pattern distribution in a user-specified permutation coding scheme.

discretization The discretization scheme to use, either 'Olivares.2012' or 'Ferri.2009'

Details

The Fisher information is a local information and complexity measure, computed based on the ordinal pattern distribution. The Fisher information is based on local gradients, hence it is sensitive to the permutation coding scheme. Options for discretization: 'Olivares.2012' or 'Ferri.2009', following Fisher Information discretization schemes in the respective publications.

Value

The normalized Fisher information measure in the range [0, 1].

Author(s)

Sebastian Sippel

References


Examples

x = arima.sim(model=list(ar = 0.3), n = 10^4)
opd = ordinal_pattern_distribution(x = x, ndemb = 6)
fis(opd = opd)
**generate_lehmerperm_matrix**

*A function to generate the Lehmer permutation ordering.*

**Description**

Generates all permutations of a given embedding dimension, ordered according to the Lehmer coding scheme.

**Usage**

```r
generate_lehmerperm_matrix(ndemb)
```

**Arguments**

- `ndemb`: The embedding dimension.

**Details**

This function converts ranks to indices and back.

**Value**

A numeric matrix that contains the Lehmer permutation pattern.

**Author(s)**

Sebastian Sippel

**References**

http://www.keithschwarz.com/interesting/code/?dir=factoradic-permutation

---

**global_complexity**

*A function to compute global information and complexity measures for time series*

**Description**

This is a high-level function that calculates global complexity measures directly from a given time series or ordinal pattern distribution.

**Usage**

```r
global_complexity(x = NA, opd = NA, ndemb)
```
Arguments

x  (OPTIONAL) If opd is not specified, a time series vector x must be specified

opd  A numeric vector that details an ordinal pattern distribution in a user-specified permutation coding scheme.

ndemb  (OPTIONAL) If x is given, the embedding dimension (ndemb) is required.

Details

This function calculates the following global measures of complexity and information:

- Number of "forbidden patterns" (cf. Amigo 2010)

Value

A named vector containing the three global complexity measures.

Author(s)

Sebastian Sippel

References


Examples

```r
x = arima.sim(model=list(ar = 0.3), n = 10^4)
global_complexity(x = x, ndemb = 6)
# or:
opd = ordinal_pattern_distribution(x = x, ndemb = 6)
global_complexity(opd = opd, ndemb = 6)
```
hellinger_distance

Distance measure between ordinal pattern distributions: Hellinger distance

Description

Compute the Hellinger Distance

Usage

hellinger_distance(p, q)

Arguments

p: An ordinal pattern distribution
q: A second ordinal pattern distribution to compare against p.

Details

This function returns a distance measure.

Value

A vector of length 1.

Author(s)

Sebastian Sippel

References

none

Examples

p = ordinal_pattern_distribution(rnorm(10000), ndemb = 5)
q = ordinal_pattern_distribution(arima.sim(model=list(ar=0.9), n=10000), ndemb = 5)
hellinger_distance(p=p, q = q)
**henon_map**

A function to generate a time series from the Henon Map

---

**Description**

Generates a time series from the Henon map

**Usage**

```r
henon_map(N, a, b, startx="rand", starty="rand", disregard_N=0)
```

**Arguments**

- **N**: length of the time series that is to be generated
- **a**: Henon map parameter a
- **b**: Henon map parameter b
- **startx**: start value in x direction. Default is to random.
- **starty**: start value in y direction. Default is to random.
- **disregard_N**: Number of values at the beginning of the series to disregard

**Value**

A vector of length N

**Author(s)**

Sebastian Sippel

**References**


**Examples**

```r
henon_map(N = 10^4, a=1.4, b=0.3)
```
HVG  

A function to compute Horizontal Visibility Graphs and associated statistics

Description

Calculates a Horizontal Visibility Graph

Usage

HVG(x, meth, maxL, rho)

Arguments

x  
A time series
meth  
A character string that describes the HVG method to use. Currently implemented: "HVG", "HVG_weighted", "LPHVG", "LPHVG_weighted".
maxL  
Maximum length of the time series.
rho  
Additional parameter

Details

Horizontal Visibility Graphs map a time series into a complex network. Following Luque, B., Lacasa, L., Ballesteros, F. and Luque, J., 2009. Horizontal visibility graphs: Exact results for random time series. Physical Review E, 80(4), p.046103. ATTENTION: This function is still in development and needs further testing!

Value

A list that contains the adjacency matrix, degree distribution, and further HVG-based statistics.

Author(s)

Sebastian Sippel

References


Examples

x = arima.sim(model=list(ar = 0.3), n = 10^2)
HVG(x, meth = "HVG", maxL = 10^9, rho = NA)
Description

Computes a normalized form of the Jensen-Shannon Divergence

Usage

jensen_shannon_divergence(p, q="unif")

Arguments

p  An ordinal pattern distribution
q  A second ordinal pattern distribution to compare against p, or a character vector q="unif" (comparison of p to uniform distribution)

Details

This function returns a distance measure.

Value

A vector of length 1.

Author(s)

Sebastian Sippel

References


Examples

p = ordinal_pattern_distribution(rnorm(10000), ndemb = 5)
q = ordinal_pattern_distribution(arima.sim(model=list(ar=0.9), n= 10000), ndemb = 5)
jensen_shannon_divergence(p = p, q = q)
limit_curves  
*Limit curves in the Entropy-Complexity plane*

**Description**
Compute the limit curves in the Entropy Complexity plane

**Usage**
```r
limit_curves(ndemb, fun = "min")
```

**Arguments**
- `ndemb`: Embedding dimension
- `fun`: Whether the upper (max) or lower (min) limit curve should be computed

**Details**
This function returns the respective limit curve.

**Value**
A list with two entries

**Author(s)**
Sebastian Sippel

**References**
none

---

logistic_map  
*A function to generate a time series from the logistic map*

**Description**
Generates a time series from the logistic map

**Usage**
```r
logistic_map(N, r, start="rand", disregard_N=0)
```
**maxd3**

**Arguments**

- **N** length of the time series that is to be generated
- **r** logistic map parameter, must be in the range [0,4]
- **start** start value. Default is to random.
- **disregard_N** Number of values at the beginning of the series to disregard

**Value**

A vector of length N

**Author(s)**

Sebastian Sippel

**References**


**Examples**

```r
logistic_map(N = 10^4, r=4)
```

---

**maxd3**  
*Maximum curve of time-causal entropy-complexity plane at ndemb=3*

**Description**

Maximum curve of time-causal entropy-complexity plane at ndemb=3

**Usage**

```r
maxd3
```

**Format**

A data frame with 494 rows and 2 columns:

- **x** x-values of minimum curve if ndemb==3
- **y** y-values of minimum curve if ndemb==3 ...

**Source**

**maxd4**

Maximum curve of time-causal entropy-complexity plane at ndemb=4

**Description**

Maximum curve of time-causal entropy-complexity plane at ndemb=4

**Usage**

maxd4

**Format**

A data frame with 2139 rows and 2 columns:

- **x** x-values of minimum curve if ndemb==4
- **y** y-values of minimum curve if ndemb==4 ...

**Source**


---

**maxd5**

Maximum curve of time-causal entropy-complexity plane at ndemb=5

**Description**

Maximum curve of time-causal entropy-complexity plane at ndemb=5

**Usage**

maxd5

**Format**

A data frame with 4151 rows and 2 columns:

- **x** x-values of minimum curve if ndemb==5
- **y** y-values of minimum curve if ndemb==5 ...

**Source**

**maxd6**

*Maximum curve of time-causal entropy-complexity plane at ndemb=6*

**Description**

Maximum curve of time-causal entropy-complexity plane at ndemb=6

**Usage**

```r
c x
```

**Format**

A data frame with 3438 rows and 2 columns:

- **x**: x-values of minimum curve if ndemb==6
- **y**: y-values of minimum curve if ndemb==6 ...

**Source**


---

**migfc**

*A function to compute Mean information gain (MIG) and Fluctuation complexity (FC)*

**Description**

Calculates MIG and FC

**Usage**

```r
migfc(x, L)
```

**Arguments**

- **x**: A time series
- **L**: word length parameter

**Details**

MIG and FC are based on a median partitioning of the time series Following Hauhs, M. and Lange, H., 2008. Classification of runoff in headwater catchments: A physical problem?. Geography Compass, 2(1), pp.235-254. ATTENTION: This function is still in development and needs further testing!
Value
A list containing MIG, FC and transition matrices.

Author(s)
Sebastian Sippel

References

Examples

```r
x = arima.sim(model=list(ar = 0.3), n = 10^4)
migfc(x, L=4)
```

Description
Minimum curve of time-causal entropy-complexity plane at ndemb=3

Usage

```r
mind3
```

Format
A data frame with 500 rows and 2 columns:

- `x` x-values of minimum curve if ndemb==3
- `y` y-values of minimum curve if ndemb==3 ...

Source
mind4

**Minimum curve of time-causal entropy-complexity plane at ndemb=4**

**Description**
Minimum curve of time-causal entropy-complexity plane at ndemb=4

**Usage**
mind4

**Format**
A data frame with 500 rows and 2 columns:
- **x** x-values of minimum curve if ndemb==4
- **y** y-values of minimum curve if ndemb==4 ...

**Source**

mind5

**Minimum curve of time-causal entropy-complexity plane at ndemb=5**

**Description**
Minimum curve of time-causal entropy-complexity plane at ndemb=5

**Usage**
mind5

**Format**
A data frame with 500 rows and 2 columns:
- **x** x-values of minimum curve if ndemb==5
- **y** y-values of minimum curve if ndemb==5 ...

**Source**
Description

Minimum curve of time-causal entropy-complexity plane at ndemb=6

Usage

mind6

Format

A data frame with 500 rows and 2 columns:

x  x-values of minimum curve if ndemb==6
y  y-values of minimum curve if ndemb==6 ...

Source


MPR_complexity

A function to compute the MPR-complexity

Description

The function computes the MPR complexity, i.e. a generalized (global) complexity measure based on the Jensen-Shannon divergence.

Usage

MPR_complexity(opd)

Arguments

opd  A numeric vector that details an ordinal pattern distribution.

Details

Generalized complexity measures combine an information measure (i.e. entropy) with the distance of the distribution from the uniform distribution ("disequilibrium"). As a global measure, MPR-complexity is insensitive to the permutation coding scheme.
**Value**

The normalized MPR complexity measure in the range [0, 1].

**Author(s)**

Sebastian Sippel

**References**


**Examples**

```r
x = arima.sim(model=list(ar = 0.3), n = 10^4)
opd = ordinal_pattern_distribution(x = x, ndemb = 6)
MPR_complexity(opd)
```

---

**nbitflips**  
*A function to compute bitflip statistics and time series*

**Description**

Computation of bitflip statistics of a time series

**Usage**

```r
nbitflips(x, ndemb)
```

**Arguments**

- **x**
  A numeric vector (e.g. a time series), from which the ordinal pattern distribution is to be calculated
- **ndemb**
  Embedding dimension of the ordinal patterns (i.e., sliding window size) for which bitflips are to be calculated. Should be chosen such as `length(x) > ndemb`

**Details**

This function returns a histogram and time series of the number of bitflips occurring in the associated ordinal patterns. NA values are allowed, and any pattern that contains at least one NA value will be ignored. WARNING: Can be slow with very long time series (n > 10^7).

**Value**

A list with two entries is returned.
 ordinal_pattern_distribution

Author(s)

Sebastian Sippel

References


Examples

x = arima.sim(model=list(ar = 0.3), n = 10^4)
nbitflips(x = x, ndemb = 6)

Description

Computation of the ordinal patterns of a time series (see e.g. Bandt and Pompe 2002)

Usage

ordinal_pattern_distribution(x, ndemb)

Arguments

x A numeric vector (e.g. a time series), from which the ordinal pattern distribution is to be calculated

ndemb Embedding dimension of the ordinal patterns (i.e. sliding window size). Should be chosen such as length(x) » ndemb

Details

This function returns the distribution of ordinal patterns using the Keller coding scheme, detailed in Physica A 356 (2005) 114-120. NA values are allowed, and any pattern that contains at least one NA value will be ignored. (Fast) C routines are used for computing ordinal patterns.

Value

A character vector of length factorial(ndemb) is returned.

Author(s)

Sebastian Sippel
**ordinal_pattern_time_series**

**References**

**Examples**

```r
x = arima.sim(model=list(ar = 0.3), n = 10^4)
ordinal_pattern_distribution(x = x, ndemb = 6)
```

**Description**
Computation of the ordinal patterns of a time series (see e.g. Bandt and Pompe 2002)

**Usage**

```r
ordinal_pattern_time_series(x, ndemb)
```

**Arguments**

- **x**
  A numeric vector (e.g. a time series), from which the ordinal pattern time series is to be calculated

- **ndemb**
  Embedding dimension of the ordinal patterns (i.e. sliding window size). Should be chosen such as length(x) » ndemb

**Details**
This function returns the distribution of ordinal patterns using the Keller coding scheme, detailed in Physica A 356 (2005) 114-120. NA values are allowed, and any pattern that contains at least one NA value will be ignored. (Fast) C routines are used for computing ordinal patterns.

**Value**
A character vector of length(x) is returned.

**Author(s)**
Sebastian Sippel

**References**
permutation_entropy

Examples

```r
x = arima.sim(model=list(ar = 0.3), n = 10^4)
ordinal_pattern_time_series(x = x, ndemb = 6)
```

permutation_entropy A function to compute the permutation entropy

Description

Computation of the permutation entropy of a time series based on its ordinal pattern distribution (see Bandt and Pompe 2002). Permutation entropy is a global information measure, hence insensitive to the permutation ordering scheme.

Usage

```r
permutation_entropy(opd)
```

Arguments

- `opd` A numeric vector that details an ordinal pattern distribution.

Details

This function calculates the permutation entropy as described in Bandt and Pompe 2002.

Value

The normalized permutation entropy as a numeric value in the range \([0,1]\).

Author(s)

Sebastian Sippel

References


Examples

```r
x = arima.sim(model=list(ar = 0.3), n = 10^4)
opd = ordinal_pattern_distribution(x = x, ndemb = 6)
permutation_entropy(opd)
```
**permutation_entropy_qlog**

*A function to compute q-log permutation entropy*

### Description

q-log permutation entropy

### Usage

```r
permutation_entropy_qlog(opd, q)
```

### Arguments

- **opd**: A numeric vector that details an ordinal pattern distribution.
- **q**: q-log parameter

### Details

This function calculates the q-log permutation entropy as described in Ribeiro et al. 2017.

### Value

The q-log permutation entropy value.

### Author(s)

Sebastian Sippel

### References


### Examples

```r
x = arima.sim(model=list(ar = 0.3), n = 10^4)
opd = ordinal_pattern_distribution(x = x, ndemb = 6)
permutation_entropy_qlog(opd = opd, q = 1)
```
permutation_entropy_Renyi

A function to compute Renyi entropy

Description

Renyi permutation entropy

Usage

permutation_entropy_Renyi(opd, alpha)

Arguments

opd A numeric vector that details an ordinal pattern distribution.
alpha alpha parameter in Renyi entropy

Details

This function calculates the Renyi entropy as described in Jauregui et al., Physica A, 498 74-85, 2018.

Value

The Renyi entropy value.

Author(s)

Sebastian Sippel

References


Examples

x = arima.sim(model=list(ar = 0.3), n = 10^4)
opd = ordinal_pattern_distribution(x = x, ndemb = 6)
permutation_entropy_Renyi(opd = opd, alpha = 0.5)
**powernoise**

A function to generate k-noise

**Description**

Generates samples of power law noise.

**Usage**

`powernoise(k, N)`

**Arguments**

- **k**: Power law scaling exponent
- **N**: number of samples to generate

**Details**

Generates samples of power law noise. The power spectrum of the signal scales as $f^{-k}$. The R function uses `fft()`, similarly to the `knoise_fft` Matlab function.

**Value**

A named list with three entries is returned. `x` - `N` x 1 vector of power law samples

**Author(s)**

Sebastian Sippel and Holger Lange

**Examples**

```r
powernoise_series = powernoise(k=2, N=10000)
```

---

**quadratic_map**

A function to generate a time series from the Quadratic map

**Description**

Generates a time series from the Quadratic map

**Usage**

`quadratic_map(N, k, start="rand", disregard_N=0)`
Arguments

- **N**: length of the time series that is to be generated
- **k**: Quadratic map parameter
- **start**: start value. Default is to random.
- **disregard_N**: Number of values at the beginning of the series to disregard

Value

A vector of length N

Author(s)

Sebastian Sippel

References


Examples

```r
quadratic_map(N = 10^4, k=1.4)
```

---

`q_complexity`  
_A function to compute q-log complexity_

Description

q-log complexity

Usage

```r
q_complexity(opd, q)
```

Arguments

- **opd**: A numeric vector that details an ordinal pattern distribution.
- **q**: q-log parameter

Details

This function calculates the q-log complexity as described in Ribeiro et al. 2017.

Value

The q-log complexity value.
**rank_to_permutation**

**Author(s)**
Sebastian Sippel

**References**

**Examples**

```r
x = arima.sim(model=list(ar = 0.3), n = 10^4)
opd = ordinal_pattern_distribution(x = x, ndemb = 6)
q_complexity(opd = opd, q = 1)
```

---

**Description**

A function to convert a "ranks-based" permutation notation to an "index-based" permutation scheme.

**Usage**

```r
rank_to_permutation(pattern, permutation.notation)
```

**Arguments**

- `pattern`: A numeric vector that denotes a permutation pattern.
- `permutation.notation`: The permutation notation that should be used. Could be "Olivares.2012" or "Keller.2005".

**Details**

This function converts ranks to indices and back.

**Value**

A numeric vector, which contains the transformed permutation.

**Author(s)**
Sebastian Sippel

**References**
### schuster_map

**A function to generate a time series from the Schuster Map**

**Description**
Generates a time series from the Schuster map

**Usage**
```
schuster_map(N, z, start="rand", disregard_N=0)
```

**Arguments**
- **N** length of the time series that is to be generated
- **z** Schuster map parameter
- **start** start value. Default is to random.
- **disregard_N** Number of values at the beginning of the series to disregard

**Value**
A vector of length N

**Author(s)**
Sebastian Sippel

**References**

**Examples**
```
schuster_map(N = 10^4, z=2)
```

---

### skew_tent_map

**A function to generate a time series from the logistic map**

**Description**
Generates a time series from the Skew-Tent map

**Usage**
```
skew_tent_map(N, a, start="rand", disregard_N=0)
```
Arguments

N  length of the time series that is to be generated  
a  Skew-Tent map parameter, must be in the range [0,1]  
start  start value. Default is to random.  
disregard_N  Number of values at the beginning of the series to disregard  

Value

A vector of length N  

Author(s)

Sebastian Sippel  

References


Examples

skew_tent_map(N = 10^4, a=0.1847)  

tent_map  A function to generate a time series from the logistic map  

Description

Generates a time series from the logistic map  

Usage

tent_map(N, mu, start="rand", disregard_N=0)  

Arguments

N  length of the time series that is to be generated  
mu  Tent map parameter, must be in the range [0,2]  
start  start value. Default is to random.  
disregard_N  Number of values at the beginning of the series to disregard  

Value

A vector of length N  

Author(s)

Sebastian Sippel
References


Examples

tent_map(N = 10^4, mu=1.8)

transformPermCoding

A function to generate a vector from an index-transformation vector from a permutation coding scheme

Description

Generates a position vector to change the ordinal pattern distribution in the default permutation coding scheme (i.e. generated by ordinal_pattern_distribution(x, ndemb)) into a user-specified coding scheme. This is a required input for the function changePermCodingOPD.

Usage

transformPermCoding(target_pattern, ndemb)

Arguments

target_pattern A numeric matrix that specifies the pattern to be transformed into the position vector.

ndemb Embedding dimension of the ordinal patterns (i.e. sliding window size). Should be chosen such as length(x) » ndemb

Details

This function returns a character vector to transform the output of ordinal_pattern_distribution (permutation coding as of Keller and Sinn, 2005) into a user-specified permutation coding scheme. For example, pattern #5 in "lehmerperm" (ndemb = 5) is given by the ranks c(0, 1, 4, 2, 3). This corresponds to pattern #41 in the (original) Keller coding scheme, as given by transformPermCoding(target_pattern = "lehmerperm", ndemb = 5)[5].

Value

A numeric vector of length factorial(ndemb), which contains the positions of the corresponding patterns in the Keller Coding scheme.

Author(s)

Sebastian Sippel
References


Examples

transformPermCoding(target_pattern = "lehmerperm", ndemb = 4)

| Turning_point | A function to compute Turning points |

Description

Calculates Turning point values.

Usage

Turning_point(x)

Arguments

x A time series

Details


Value

Turning point value

Author(s)

Sebastian Sippel

References

Tarnopolski et al. (2016), Physica A 461, 662-673.

Examples

x = arima.sim(model=list(ar = 0.3), n = 10^4)
Turning_point(x)
**weighted_ordinal_pattern_distribution**

*A function to compute weighted ordinal pattern statistics*

**Description**

Computation of weighted ordinal patterns of a time series. Weights can be generated by a user-specified function (e.g. variance-weighted, see Fadlallah et al 2013).

**Usage**

```r
weighted_ordinal_pattern_distribution(x, ndemb)
```

**Arguments**

- `x` A numeric vector (e.g. a time series), from which the weighted ordinal pattern distribution is to be calculated
- `ndemb` Embedding dimension of the ordinal patterns (i.e. sliding window size). Should be chosen such as length(x) \(\geq\) ndemb

**Details**

This function returns the distribution of weighted ordinal patterns using the Keller coding scheme, detailed in Physica A 356 (2005) 114-120. NA values are allowed. The function uses old and slow R routines and is only maintained for comparability. For faster routines, see `weighted_ordinal_pattern_distribution`.

**Value**

A character vector of length factorial(ndemb) is returned.

**Author(s)**

Sebastian Sippel

**References**


**See Also**

`weighted_ordinal_pattern_distribution`

**Examples**

```r
x = arima.sim(model=list(ar = 0.3), n = 10^4)
weighted_ordinal_pattern_distribution(x = x, ndemb = 6)
```
Index

*Topic datasets
  maxd3, 13
  maxd4, 14
  maxd5, 14
  maxd6, 15
  mind3, 16
  mind4, 17
  mind5, 17
  mind6, 18

Abbe, 2
adjust_pattern, 3

complexity_Renyi, 4

fis, 5

generate_lehmerperm_matrix, 6
global_complexity, 6

hellinger_distance, 8
henon_map, 9
HVG, 10

jensen_shannon_divergence, 11

limit_curves, 12
logistic_map, 12

maxd3, 13
maxd4, 14
maxd5, 14
maxd6, 15
migfc, 15
mind3, 16
mind4, 17
mind5, 17
mind6, 18
MPR_complexity, 18

nbitflips, 19

ordinal_pattern_distribution, 20
ordinal_pattern_time_series, 21

permutation_entropy, 22
permutation_entropy_qlog, 23
permutation_entropy_Renyi, 24
powernoise, 25

q_complexity, 26
quadratic_map, 25

rank_to_permutation, 27

schuster_map, 28
skew_tent_map, 28

tent_map, 29
transformPermCoding, 30
Turning_point, 31

weighted_ordinal_pattern_distribution, 32, 32