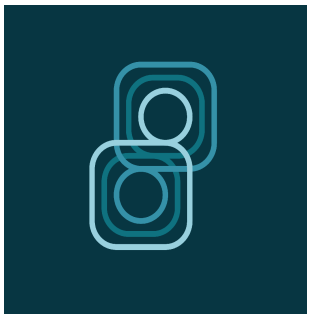
THE ENTROPYHUB GUIDE

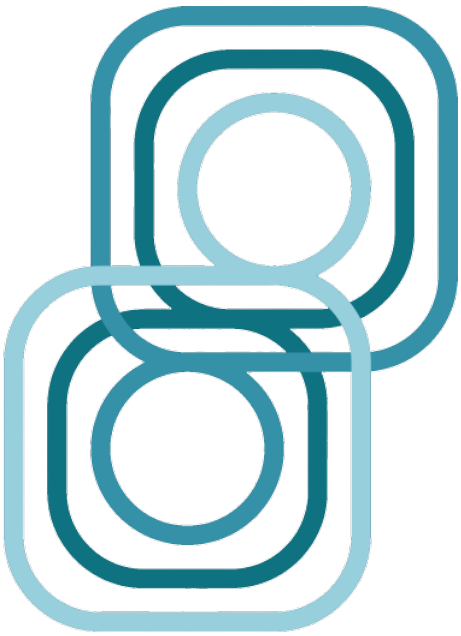
A user manual for the EntropyHub toolkit

Matthew W. Flood

www.EntropyHub.xyz

v.2.0 (2024)





– I prefer to take the names of important scientific quantities from ancient languages, so they may be the same in all living languages. I therefore propose to call ***entropy*** *the quantity (S) of a body from the Greek word for transformation:* $\eta\ \tau\rho o\pi\eta$

Rudolf Clausius

– Quantities of the form $H = -\Sigma\ p_i\ Log\ p_i$ *play a central role in information theory as measures of information, choice and uncertainty. The form of* H *will be recognized as that of* ***entropy*** *as defined in certain formulations of statistical mechanics where* p_i *is the probability of a system being in cell* i *of its phase space. H is then, for example, the H in Boltzmann's famous H theorem. We shall call* $H = -\Sigma\ p_i\ Log\ p_i$ *the* ***entropy*** *of the set of probabilities* $p_1, \ldots, p_n$.

Claude Shannon

– The fact that you can remember yesterday but not tomorrow is because of ***entropy****. The fact that you're always born young and then you grow older, and not the other way around like Benjamin Button - it's all because of* ***entropy****. So I think that* ***entropy*** *is underappreciated as something that has a crucial role in how we go through life.*

Sean M.Carroll

Preface

The concept of entropy has its origins in classical physics under the second law of thermodynamics, a law considered to underpin our fundamental understanding of time in physics. Attempting to analyse the analog world around us requires that we measure time in discrete steps, but doing so compromises our ability to measure entropy accurately. Since the introduction of approximate entropy by Pincus three decades ago[1], the use of information theoretic entropy measures to estimate the complexity, randomness or regularity of time series data has become ubiquitous in many research domains (Fig. 1a). Applications of entropy are ever-increasing (Fig. 1b), as are the number of new entropies that aim to estimate entropy with greater accuracy, less sensitivity to data length, amplitude fluctuations, etc. (see Ribiero et al.[2])

Although many functions for estimating these entropies can be found in various corners of the internet, there is currently no toolkit to perform entropic time-series analysis at the

[1] Steven M. Pincus,
Approximate entropy as a measure of system complexity,
Proceedings of the National Academy of Sciences (1991); 88.6: 2297-2301

[2] Ribeiro M, Henriques T, Castro L, Souto A, Antunes L, Costa-Santos C, Teixeira A.,
The Entropy Universe,
Entropy (2021); 23(2):222

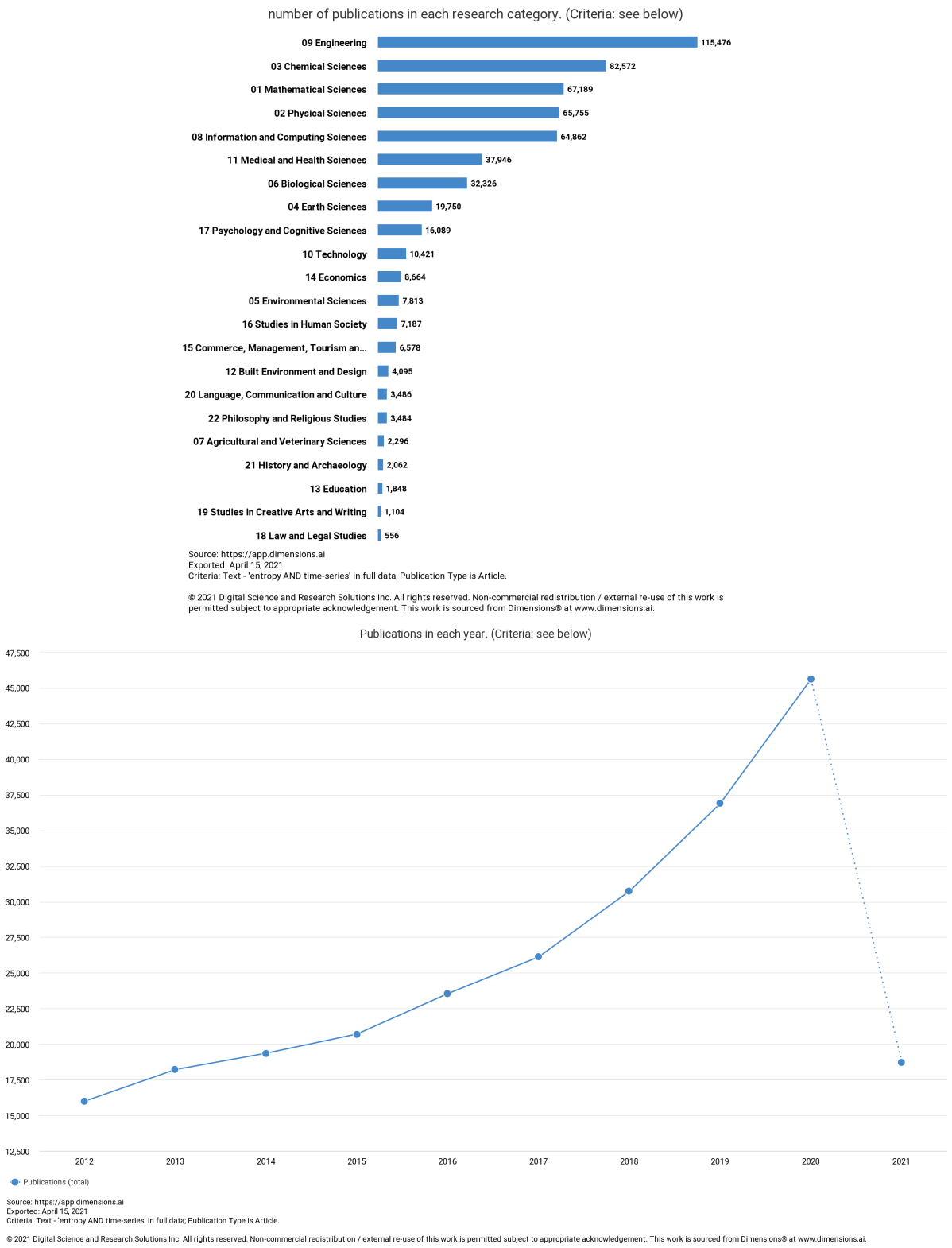


Figure 1: Research domains and the number of publications each year featuring the terms 'Entropy' AND 'Time-Series' *from 2012-2021. (Source: Dimensions.ai)*

command line with reliable code, extensive documentation and consistent syntax, that is also accessible in multiple programming languages. Hence, the goal of EntropyHub is to integrate the many established entropy methods into one package that is available for users of Python, MatLab and Julia.

EntropyHub features multiscale variants of all base and cross-entropy methods, (including composite, refined and hierarchical multiscale approaches), in addition to bidimensional entropies for 2D matrix analysis. As the scientific community develops novel entropic measures, efforts will be made to incorporate them in later versions of the package.

Matthew W. Flood (2021),
EntropyHub: An Open-Source Toolkit for Entropic Time Series Analysis,
PLoS ONE 16(11):e0259448
DOI: 10.1371/journal.pone.0259448
www.EntropyHub.xyz

If you find this package useful, please consider starring it on GitHub, MatLab File Exchange, PyPI or Julia Packages. This helps us to gauge user satisfaction.

Thank you for using EntropyHub,

Matt
info@entropyhub.xyz

Table of Contents

1 Introduction **1**
- 1.1 **Updates** 4
  - 1.1.1 **v2.0** - [April 2024] 4
  - 1.1.2 **v1.0** - [March 2024] 5
- 1.2 **Contact** 7

2 Installation **8**
- 2.1 **MatLab** 8
- 2.2 **Python** 15
- 2.3 **Julia** 18

3 Functions **20**
- 3.1 Base Entropy Functions 21
  - 3.1.1 ApEn: Approximate Entropy 21
  - 3.1.2 SampEn: Sample Entropy 22
  - 3.1.3 FuzzEn: Fuzzy Entropy 23
  - 3.1.4 K2En: Kolmogorov Entropy 27
  - 3.1.5 PermEn: Permutation Entropy 28
  - 3.1.6 CondEn: *corrected* Conditional Entropy 29
  - 3.1.7 DistEn: Distribution Entropy 30
  - 3.1.8 SpecEn: Spectral Entropy 31
  - 3.1.9 DispEn: Dispersion Entropy 32
  - 3.1.10 SyDyEn: Symbolic Dynamic Entropy 34
  - 3.1.11 IncrEn: Increment Entropy 36
  - 3.1.12 CoSiEn: Cosine Similarity Entropy 37
  - 3.1.13 PhasEn: Phase Entropy 38
  - 3.1.14 SlopEn: Slope Entropy 39
  - 3.1.15 BubbEn: Bubble Entropy 40
  - 3.1.16 GridEn: Gridded Distribution Entropy 41
  - 3.1.17 EnofEn: Entropy of Entropy 42
  - 3.1.18 AttnEn: Attention Entropy 43
  - 3.1.19 DivEn: Diversity Entropy 44
  - 3.1.20 RangEn: Range Entropy 45
- 3.2 Cross-Entropy Functions 46
  - 3.2.1 XApEn: Cross-Approximate Entropy 46
  - 3.2.2 XSampEn: Cross-Sample Entropy 47
  - 3.2.3 XFuzzEn: Cross-Fuzzy Entropy 48
  - 3.2.4 XK2En: Cross-Kolmogorov Entropy 52
  - 3.2.5 XPermEn: Cross-Permutation Entropy 53
  - 3.2.6 XCondEn: Cross-Conditional Entropy 54
  - 3.2.7 XDistEn: Cross-Distribution Entropy 55

3.2.8 XSpecEn: Cross-Spectral Entropy 56
3.3 Multivariate Entropy Functions 57
3.3.1 MvSampEn: Multivariate Sample Entropy 58
3.3.2 MvFuzzEn: Multivariate Fuzzy Entropy 59
3.3.3 MvDispEn: Multivariate Dispersion Entropy 64
3.3.4 MvPermEn: Multivariate Permutation Entropy 66
3.3.5 MvCoSiEn: Multivariate Cosine Similarity Entropy 68
3.4 Multiscale Entropy Functions 69
3.4.1 MSobject: Multiscale Entropy Object 70
3.4.2 MSEn: Multiscale Entropy 71
3.4.3 cMSEn: Composite & Refined-Composite Multiscale Entropy 73
3.4.4 rMSEn: Refined Multiscale Entropy 75
3.4.5 hMSEn: Hierarchical Multiscale Entropy 77
3.5 Multiscale Cross-Entropy Functions 79
3.5.1 MSobject: Multiscale Entropy Object 81
3.5.2 XMSEn: Multiscale Cross-Entropy 82
3.5.3 cXMSEn: Composite & Refined-Composite Multiscale Cross-Entropy 84
3.5.4 rXMSEn: Refined Multiscale Cross-Entropy 86
3.5.5 hXMSEn: Hierarchical Multiscale Cross-Entropy 88
3.6 Multivariate Multiscale Entropy Functions 90
3.6.1 MvMSEn: Multivariate Multiscale Entropy 91
3.6.2 cMvMSEn: Composite & Refined-Composite Multivariate Multiscale Entropy 92
3.7 Bidimensional Entropy Functions 93
3.7.1 SampEn2D: Bidimensional Sample Entropy 94
3.7.2 FuzzEn2D: Bidimensional Fuzzy Entropy 95
3.7.3 DistEn2D: Bidimensional Distribution Entropy 99
3.7.4 DispEn2D: Bidimensional Dispersion Entropy 100
3.7.5 PermEn2D: Bidimensional Permutation Entropy 102
3.7.6 EspEn2D: Bidimensional Espinosa Entropy 104
3.8 Other Functions 105
3.8.1 ExampleData(): Import Example Datasets 105
3.8.2 WindowData(): Data Windowing Tool 106

4 Examples 107
4.1 MatLab: 109
4.1.1 Example 1: Sample Entropy 109
4.1.2 Example 2: (Fine-Grained) Permutation Entropy 110
4.1.3 Example 3: Phase Entropy w/ Poincaré plot 111
4.1.4 Example 4: Cross-Distribution Entropy w/ Different Binning Methods 113
4.1.5 Example 5: Multiscale Entropy Object [MSobject()] 114
4.1.6 Example 6: Multiscale [Increment] Entropy 115
4.1.7 Example 7: Refined Multiscale [Sample] Entropy 116
4.1.8 Example 8: Composite Multiscale Cross-[Approximate] Entropy 117
4.1.9 Example 9: Hierarchical Multiscale *corrected* Cross-[Conditional] Entropy 118
4.1.10 Example 10: Bidimensional Fuzzy Entropy 120

4.1.11 Example 11: Multivariate Dispersion Entropy 121
4.1.12 Example 12: [Generalized] Refined-composite Multivariate Multiscale Fuzzy Entropy 122
4.1.13 Example 13: Windowing Data with WindowData() 124
4.2 Python: 127
4.2.1 Example 1: Sample Entropy 128
4.2.2 Example 2: (Fine-Grained) Permutation Entropy 129
4.2.3 Example 3: Phase Entropy w/ Poincaré plot 130
4.2.4 Example 4: Cross-Distribution Entropy w/ Different Binning Methods 132
4.2.5 Example 5: Multiscale Entropy Object [MSobject()] 133
4.2.6 Example 6: Multiscale [Increment] Entropy 134
4.2.7 Example 7: Refined Multiscale [Sample] Entropy 135
4.2.8 Example 8: Composite Multiscale Cross-[Approximate] Entropy 136
4.2.9 Example 9: Hierarchical Multiscale *corrected* Cross-[Conditional] Entropy 137
4.2.10 Example 10: Bidimensional Fuzzy Entropy 139
4.2.11 Example 11: Multivariate Dispersion Entropy 140
4.2.12 Example 12: [Generalized] Refined-composite Multivariate Multiscale Fuzzy Entropy 141
4.2.13 Example 13: Windowing Data with WindowData() 143
4.3 Julia: 146
4.3.1 Example 1: Sample Entropy 147
4.3.2 Example 2: (Fine-Grained) Permutation Entropy 148
4.3.3 Example 3: Phase Entropy w/ Poincaré plot 149
4.3.4 Example 4: Cross-Distribution Entropy w/ Different Binning Methods 151
4.3.5 Example 5: Multiscale Entropy Object [MSobject()] 152
4.3.6 Example 6: Multiscale [Increment] Entropy 153
4.3.7 Example 7: Refined Multiscale [Sample] Entropy 154
4.3.8 Example 8: Composite Multiscale Cross-[Approximate] Entropy 155
4.3.9 Example 9: Hierarchical Multiscale *corrected* Cross-[Conditional] Entropy 156
4.3.10 Example 10: Bidimensional Fuzzy Entropy 158
4.3.11 Example 11: Multivariate Dispersion Entropy 159
4.3.12 Example 12: [Generalized] Refined-composite Multivariate Multiscale Fuzzy Entropy 160
4.3.13 Example 13: Windowing Data with WindowData() 162

5 References 165

6 Glossary of Function Syntax 171

1 Introduction

IMPORTANT NOTE

It is important to clarify at the outset that the term *entropy* henceforth described refers to entropy in the context of dynamical systems, probability theory and information theory as defined by Shannon[a] , and not thermodynamic or other entropies from classical physics.

[a]Claude E. Shannon,
A Mathematical Theory of Communication
Bell System Technical Journal (1948), 27 (3): 379–423.

EntropyHub functions fall into five categories:

Base	functions for estimating the entropy of a single univariate data sequence.
Cross	functions for estimating the entropy between two univariate data sequences.
Bidimensional	functions for estimating the entropy of a two-dimensional univariate matrix.
Multiscale	functions for estimating the multiscale entropy of a single univariate data sequences using any of the Base entropy functions.
Multiscale Cross-	functions for estimating the multiscale entropy between two univariate data sequences using any of the Cross-entropy functions.

[See *Table 1.1* for a list of all functions]

While each function has its own unique keyword arguments, there are several keyword arguments (also known as Name/Value pairs in MatLab) common to most `Base, Cross` and `Bidimensional` entropies. These are:

`m`	embedding dimension
`tau`	time delay
`Logx`	base of the logarithm in Shannon's formula for entropy. (this argument allows the entropy to be estimated in bits (base 2), nats (base e), dits (base 10), or whatever the user specifies)
`Norm`	normalisation of the entropy value as outlined in the source literature for that particular function.

All `Multiscale` and `Multiscale Cross`–entropy functions keyword arguments are identical.

One of the advantages of EntropyHub is the variety of keyword arguments available for many functions. For example, by specifying the **`Typex`** keyword argument when calling `PermEn`, one can calculate the edge, weighted, modified, amplitude-aware, fine-grained or uniform-quantization variants of permutation entropy, in addition to the original defined by Bandt and Pope [9]. Similarly, one can employ different fuzzy functions to transform state vector distances when calculating fuzzy entropy (`FuzzEn`) by specifying the **`Fx`** keyword argument. This ability to augment various parameters at the command line enables more advanced entropy methods to be performed with ease.

IMPORTANT NOTE

Although each function is complete with default arguments, blindly analysing time series data using these arguments is **strongly discouraged**.
Inferring meaning about time series from entropy estimates is only valid when the parameters used accurately capture the underlying dynamics of the data.

Each function has a helpful description of its usage in the docstrings, explaining input parameters, outputs values and references to relevant source literature. To read the docstrings of a particular function, type:

MatLab	`help` *`function-name`*	e.g. `help PermEn`
Python	`help(`*`EntropyHub.function-name`*`)`	e.g. `help EntropyHub.PermEn`
Julia	`?` *`function-name`*	e.g. `?  PermEn()`

BONUS

While the majority of multiscale and multiscale-cross functions available through EntropyHub have been previously published, options are available to call new multiscale variants, such as *multiscale cross-spectral entropy*.

EntropyHub Function List			
Base Entropy	*Function*	*Cross-Entropy*	*Function*
Approximate Entropy	ApEn	Cross Sample Entropy	XSampEn
Sample Entropy	SampEn	Cross Approximate Entropy	XApEn
Fuzzy Entropy	FuzzEn	Cross Fuzzy Entropy	XFuzzEn
Kolmogorov Entropy	K2En	Cross Permutation Entropy	XPermEn
Permutation Entropy	PermEn	Cross Conditional Entropy	XCondEn
Conditional Entropy	CondEn	Cross Distribution Entropy	XDistEn
Distribution Entropy	DistEn	Cross Spectral Entropy	XSpecEn
Spectral Entropy	SpecEn	Cross Kolmogorov Entropy	XK2En
Dispersion Entropy	DispEn		
Symbolic Dynamic Entropy	SyDyEn	*Multivariate Entropy*	*Function*
Cosine Similarity Entropy	CoSiEn	Multivariate Sample Entropy	MvSampEn
Phase Entropy	PhasEn	Multivariate Fuzzy Entropy	MvFuzzEn
Slope Entropy	SlopEn	Multivariate Permutation Entropy	MvPermEn
Bubble Entropy	BubbEn	Multivariate Dispersion Entropy	MvDispEn
Gridded Distribution Entropy	GridEn	Multivariate Cosine Similarity Entropy	MvCoSiEn
Increment Entropy	IncrEn		
Entropy of Entropy	EnofEn	*Bidimensional Entropy*	*Function*
Attention Entropy	AttnEn	2D Sample Entropy	SampEn2D
Range Entropy	RangEn	2D Fuzzy Entropy	FuzzEn2D
Diversity Entropy	DivEn	2D Distribution Entropy	DistEn2D
		2D Dispersion Entropy	DispEn2D
		2D Permutation Entropy	PermEn2D
		2D Espinosa Entropy	EspEn2D
Multiscale Entropy	*Function*	*Multiscale Cross-Entropy*	*Function*
Multiscale Entropy	MSEn	Multiscale Cross-Entropy	XMSEn
Composite + Refined-Composite Multiscale Entropy	CMSEn	Composite + Refined-Composite Multiscale Cross-Entropy	cXMSEn
Refined Multiscale Entropy	rMSEn	Refined Multiscale Cross-Entropy	rXMSEn
Hierarchical Multiscale Entropy	hMSEn	Hierarchical Multiscale Cross-Entropy	hXMSEn
Multivariate Multiscale Entropy	*Function*	*Other Functions*	*Function*
Multivariate Multiscale Entropy	MvMSEn	Example Dataset Importer	ExampleData
Composite + Refined-Composite Multivariate Multiscale Entropy	cMvMSEn	Window Data Tool	WindowData

Table 1.1: List of functions in the EntropyHub toolkit.

1.1 Updates

1.1.1 v2.0 - [April 2024]

Since the introduction of multivariate entropy methods over a decade ago, the utilization of multivariate and multivariate multiscale entropy methods has seen a notable increase in recent years. Systems comprising multiple dynamically related components are ubiquitous in many research fields and multivariate entropies provide a powerful method for estimating the complexity of such systems.
With EntropyHub v2.0, you can now apply many multivariate and multivariate multiscale methods with the ease, robust functionality and extensive documentation for which EntropyHub is acclaimed.

\+ **New multivariate methods**
Five new multivariate entropy functions incorporating several method-specific variations.

Multivariate Sample Entropy - 3.3.1 **`MvSampEn()`**
Based on methods presented in [79], [80]

Multivariate Fuzzy Entropy - 3.3.2 **`MvFuzzEn()`**
++ many fuzzy functions, [81]

Multivariate Dispersion Entropy - 3.3.3 **`MvDispEn()`**
++ many symbolic sequence transforms, [82]

Multivariate Cosine Similarity Entropy - 3.3.5 **`MvCoSiEn()`**
Based on methods presented in [78],

Multivariate Permutation Entropy - 3.3.4 **`MvPermEn()`**
++ amplitude-aware, edge, phase, weighted and modified variants

\+ **New multivariate multiscale methods**
Two new multivariate multiscale entropy functions:

Multivariate Multiscale Entropy - 3.6.1 **`MvMSEn()`**
++ coarse, modified and generalized graining procedures

Composite and Refined-composite Multivariate Multiscale Entropy - 3.6.2 **`cMvMSEn()`**

\+ **Extra signal processing tools**

WindowData() is a new function that allows users to segment data (univariate or multivariate time series) into windows with/without overlapping samples! This allows users to calculate entropy on subsequences of their data to perform analyses with greater time resolution.

Other little fixes...

✓ **Docs edits**
Examples in the www.EntropyHub.xyz documentation were updated to match the latest package syntax.

1.1.2 v1.0 - [March 2024]

EntropyHub is continuously growing to incorporate the lastest developments in the scientific literature. This new major release (v1.0) reflects that with many new functions and features to provide you with a versatile environment that makes complex entropy methods easy to implement.
The following list summarizes some of the main updates.

+ **New entropy methods**
  Two new base entropy functions (and their multiscale versions) have been added:

  Diversity Entropy - 3.1.19 **`DivEn()`**
  A method that employs cosine similarity to measure similarity between delay vectors.[43]

  Range Entropy - 3.1.20 **`RangEn()`**
  A method that uses the rescaled range to normalize distances between delay vectors, thereby reducing the ambiguity in selecting the threshold tolerance in ApEn or SampEn.[45]

+ **New fuzzy membership functions**
  Several new fuzzy membership functions have been added to **`FuzzEn`** (3.1.3), **`XFuzzEn`** (3.2.3) and **`FuzzEn2D`** (3.7.2) to provide more options for mapping the degree of similarity between embedding vectors. These include

  - `trapezoidal`
  - `triangular`
  - `gaussian`
  - `constgaussian`
  - `z-shaped`
  - `bell-shaped`

  Further info on these membership functions can be found here.

+ ***Phase* Permutation Entropy**
  A new variant - ***phase*** permutation entropy - has been added to **`PermEn`** (3.1.5). This method employs a hilbert transformation of the data sequence, based on the methods outlined here.

+ **Cross-Entropy with different length sequences**
  EntropyHub now allows for cross-entropy (and multiscale cross-entropy) estimation with different length signals (except **`XCondEn`** and **`XPermEn`**). As a result, the new cross-entropy functions require a separate input for each sequence (**`Sig1, Sig2`**).

+ **Refined-Composite Multiscale Fuzzy Entropy**
  In addition to the refined-composite multiscale sample entropy that was available in earlier versions (**`cMSEn`** - 3.4.3), now one can estimate the refined-composite multiscale fuzzy entropy based on the method outlined here. What's more, refined-composite multicale cross-fuzzy entropy is also available (**`cXMSEn`** - 3.5.3), and both can be estimated using any of the fuzzy membership functions in **`FuzzEn`** (3.1.3) or **`XFuzzEn`** (3.2.3).
  Note: refined-composite multiscale sample entropy uses moving averaging to perform graining (analogous to the `modified` graining method in **`MSEn()`**), but

refined-composite multiscale fuzzy entropy uses the moving variance (analogous to the `generalized` graining method in **`MSEn()`**)

+ **Generalized Multiscale Entropy**
  Generaized multiscale entropy and generalized multiscale cross-entropy can now be estimated. Just choose the **`"generalized"`** as the graining procedure in **`MSEn`** (3.4.2) or **`XMSEn`** (3.5.2).

+ **Variance of sample entropy**
  Based on the method outlined by Lake et al. [3], it is now possible to obtain a measure of the variance in the sample entropy estimate. This is achieved by approximating the number of overlapping embedding vectors. To do so, just set the parameter **`Vcp==true`** in **`SampEn`** (3.1.2) and **`XSampEn`** (3.2.2), but **note that doing so requires a lot of computer memory.**

Several little bugs and inconsistencies have also been fixed in this release. We want to thank all who have identified and alerted us to these bugs. Most of these bugs have been noted via the GitHub issues portal.

Bug fixes

- ✓ The **`DispEn2D`** (3.7.4) function in python has now fixed this issue.
- ✓ The type hint for **`FuzzEn`** (3.1.3) in python has been updated as requested.
- ✓ Compatbility issues with ***EntropyHub.jl*** are now resolved.
- ✓ A bug in the **`K2En`** (3.1.4) python function led to incorrect entropy estimates for data sequences with many equal values. This has been corrected.

Other Changes

- ↑ The **`equal`** method for discretizing data in **`DispEn`** (3.1.9) and **`DispEn2D`** (3.7.4) has been updated to be consistent across Python, MatLab and Julia. This is unlikely to have impacted any users previously.
- ↑ The zeroth dimension (**`m=0`**) estimate of **`ApEn`** (3.1.1) and **`XApEn`** (3.2.1) has been changed to $-\Phi_1$.
- ↑ The default radius threshold distance for **`XApEn`**(3.2.1), **`XSampEn`** (3.2.2) and **`XK2En`** (3.2.4) has been changed to use the pooled standard deviation [i.e. $0.2{*}SD_{pooled}(X,Y)$].
- ↑ The default number of points used in the Fourier spectrum transform in XSpecEn has been changed to **`2*max(length(Sig1),length(Sig2)) + 1`**.
- ↑ The default radius parameter value for **`FuzzEn2D`** (3.7.2) has been changed to (0.2*std(Mat),2).

1.2 Contact

EntropyHub is linked to many online resources that provide further information about the toolkit and installation files. In addition to this, users can directly contact the EntropyHub developers to seek help, report bugs, or suggest features to improve the toolkit. The following 1.2 provides a list of email addresses and links to EntropyHub resources.

Online Resources	
EntropyHub website	www.EntropyHub.xyz *or alternatively* MattWillFlood.github.io/EntropyHub
EntropyHub Julia Website	MattWillFlood.github.io/EntropyHub.jl
EntropyHub GitHub Repo	github.com/MattWillFlood/EntropyHub
MatLab: File Exchange	www.mathworks.com/matlabcentral/fileexchange/94185-entropyhub
Python: PyPI	pypi.org/project/EntropyHub/
Julia: General Registry	juliahub.com/ui/Packages/EntropyHub/npy5E/ Julia Registry (GitHub)
Email Addresses	
General inquiries	info@entropyhub.xyz
Seeking help	help@entropyhub.xyz
Report bugs or errors	fix@entropyhub.xyz

Table 1.2: EntropyHub resources and contact details.

LET'S GET STARTED!

2 Installation

Stable releases of EntropyHub are available from the default package manager for MatLab (File Exchange), Python (PyPi) and Julia (Julia Packages), while the latest version of EntropyHub can be downloaded or cloned from the GitHub repository.

2.1 MatLab

System Requirements

There are two additional MatLab toolboxes required to exploit the *full* functionality of the EntropyHub toolkit:

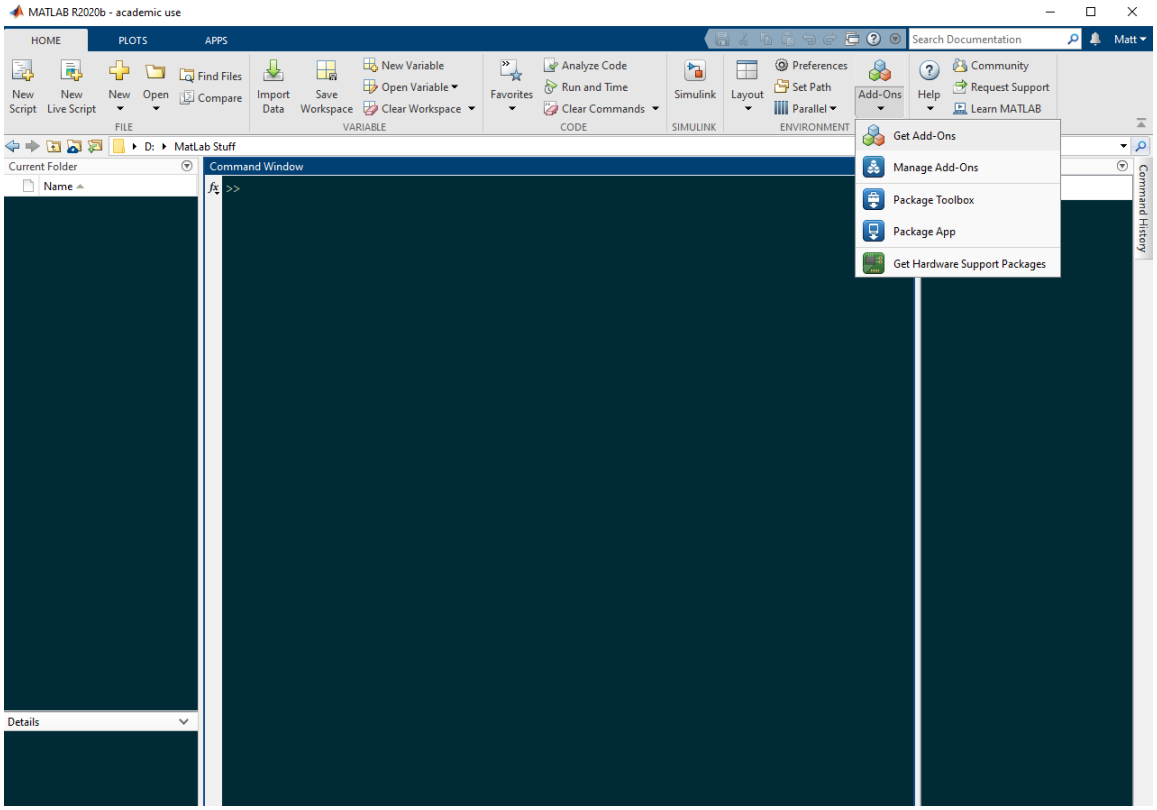
Signal Processing Toolbox

Statistics and Machine Learning Toolbox

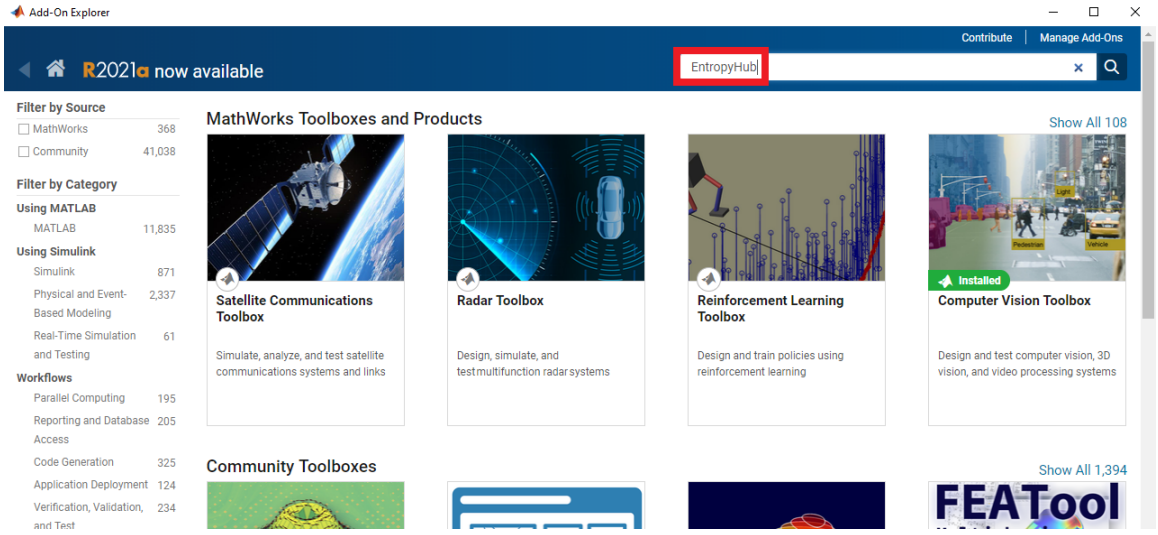
however, most functions will work without these toolboxes.
EntropyHub is intended for use with MatLab versions >= 2016a. In some cases the toolkit may work on versions 2015a and 2015b. However, it is not recommended to install on MatLab versions older than 2016 and should be done so with caution.
There are 3 ways to install EntropyHub for Matlab. Method 1 is the most straightforward.

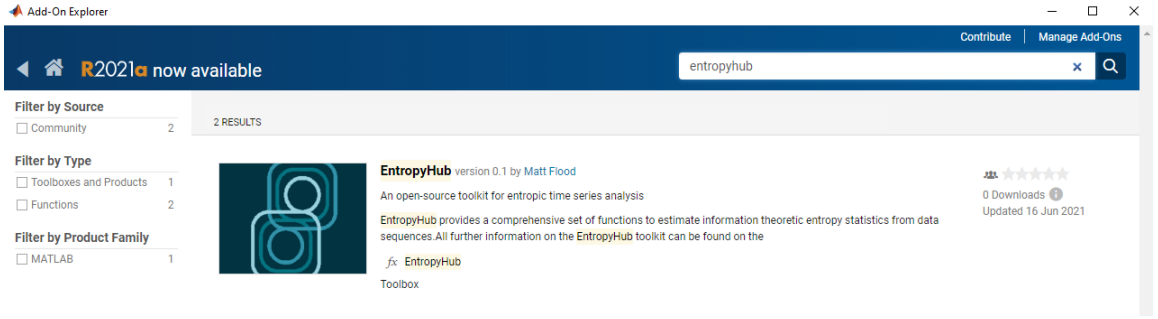
Method 1.

1. In MatLab, click the 'Add-Ons' button in the HOME tab. This should open the MatLab Add-On Explorer.

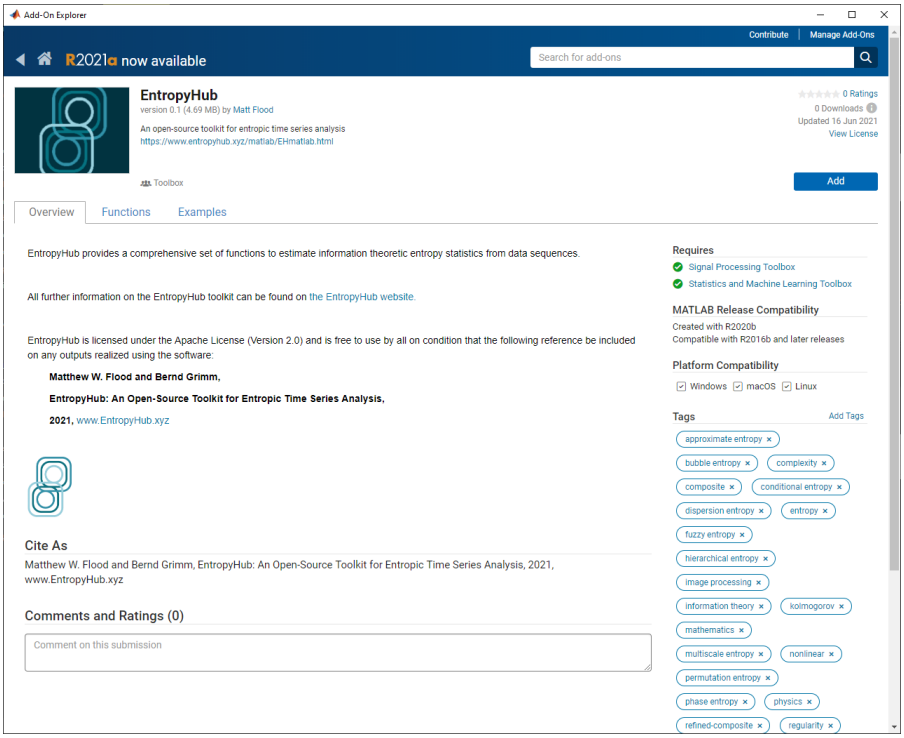


2. In the Add-On Explorer, search for 'EntropyHub'.

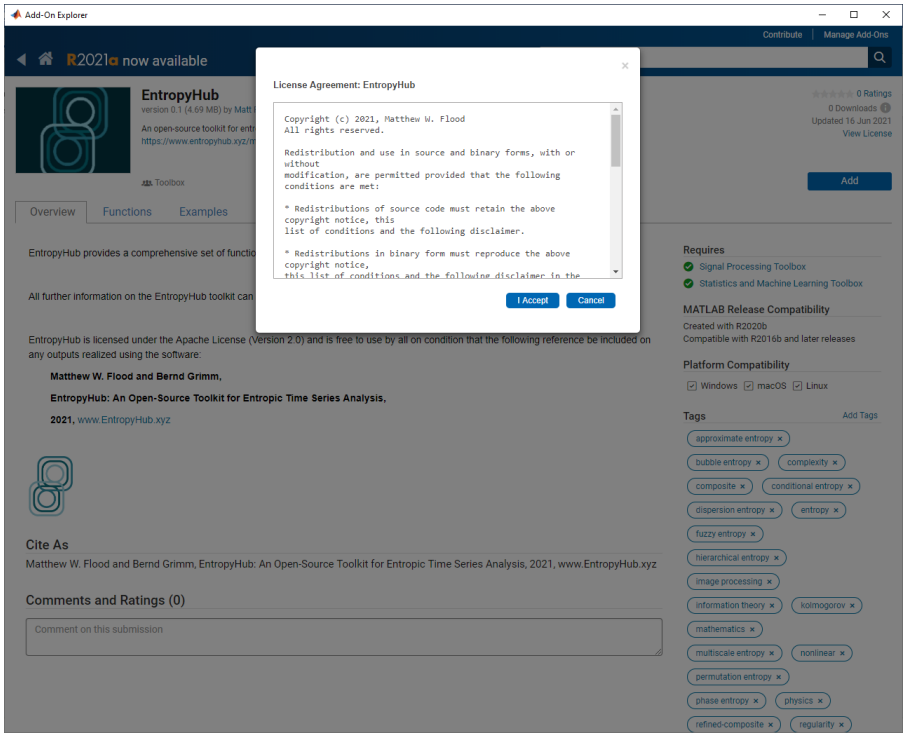




3. Open the link to EntropyHub and click the 'Add' button in the top right corner.

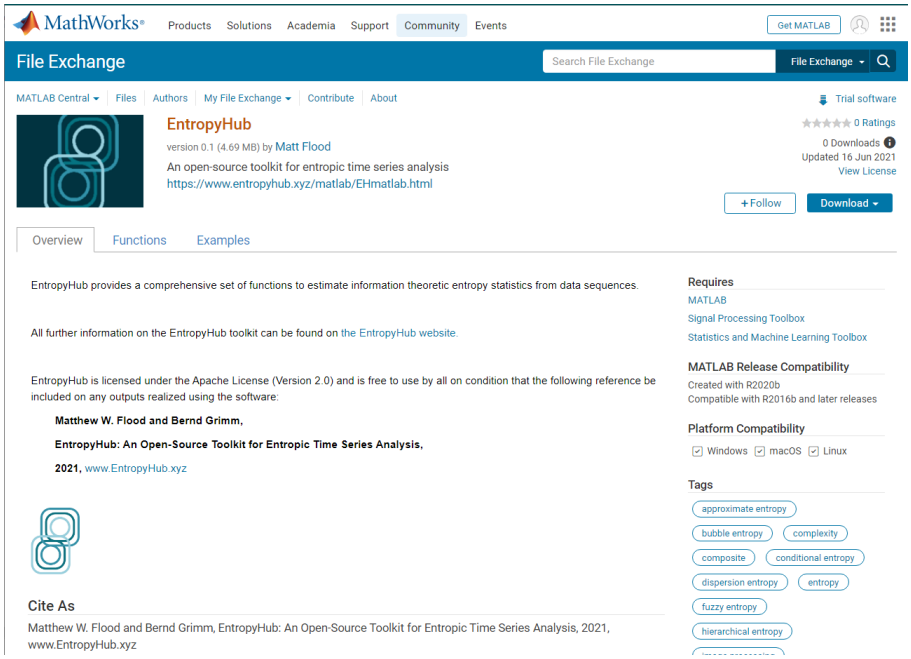


You will be asked to accept the License Agreement prior to installation.



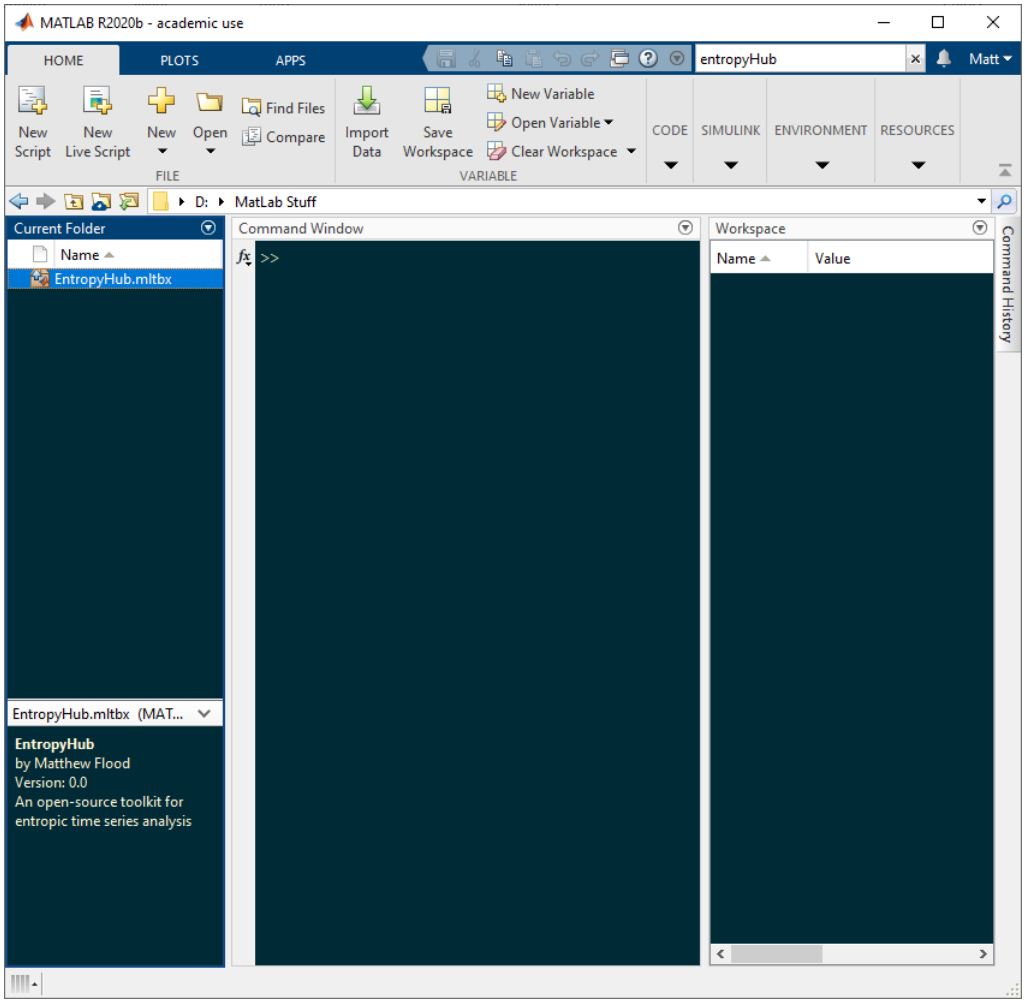
Method 2.

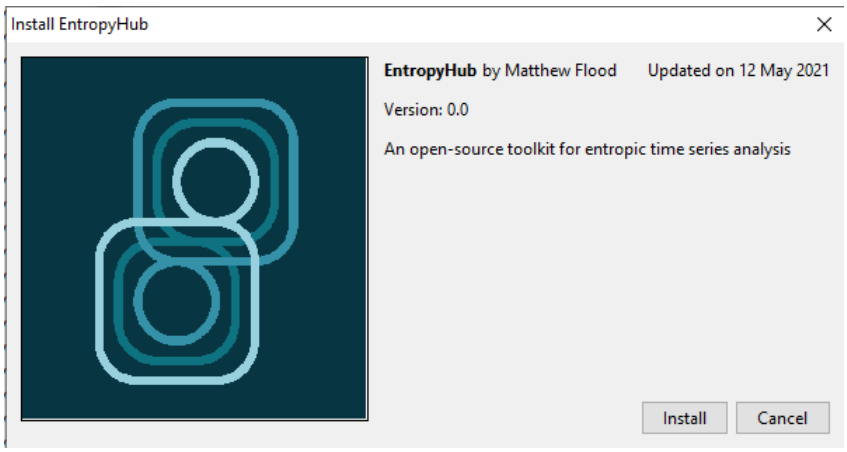
1. Visit the EntropyHub File Exchange page.
   Note: you need to be logged in to your MathWorks account to continue.



2. Download the toolbox file (EntropyHub.mltbx) by clicking 'Toolbox' in the drop-down menu under the 'Download' button on the right hand side.

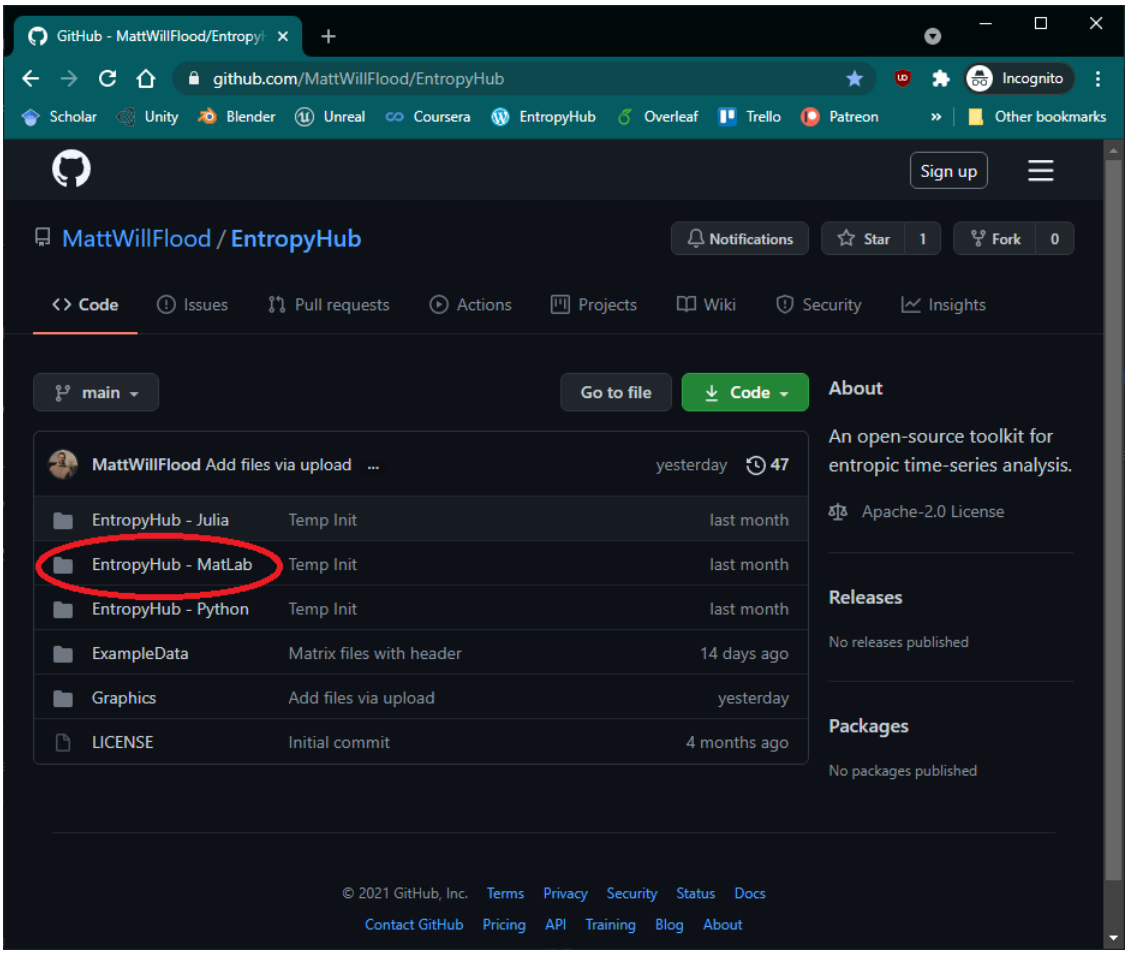
3. In MatLab, navigate the current folder to the directory where the EntropyHub.mltbx file is saved. Open the file and click install.



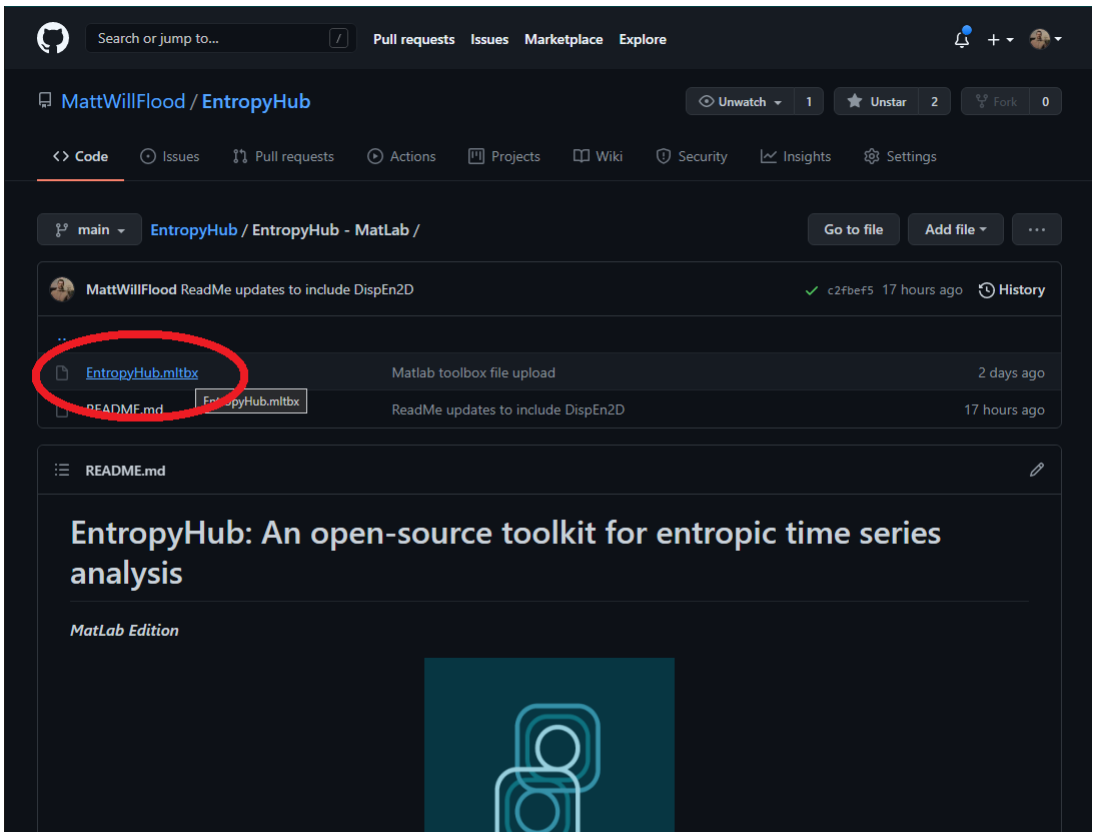


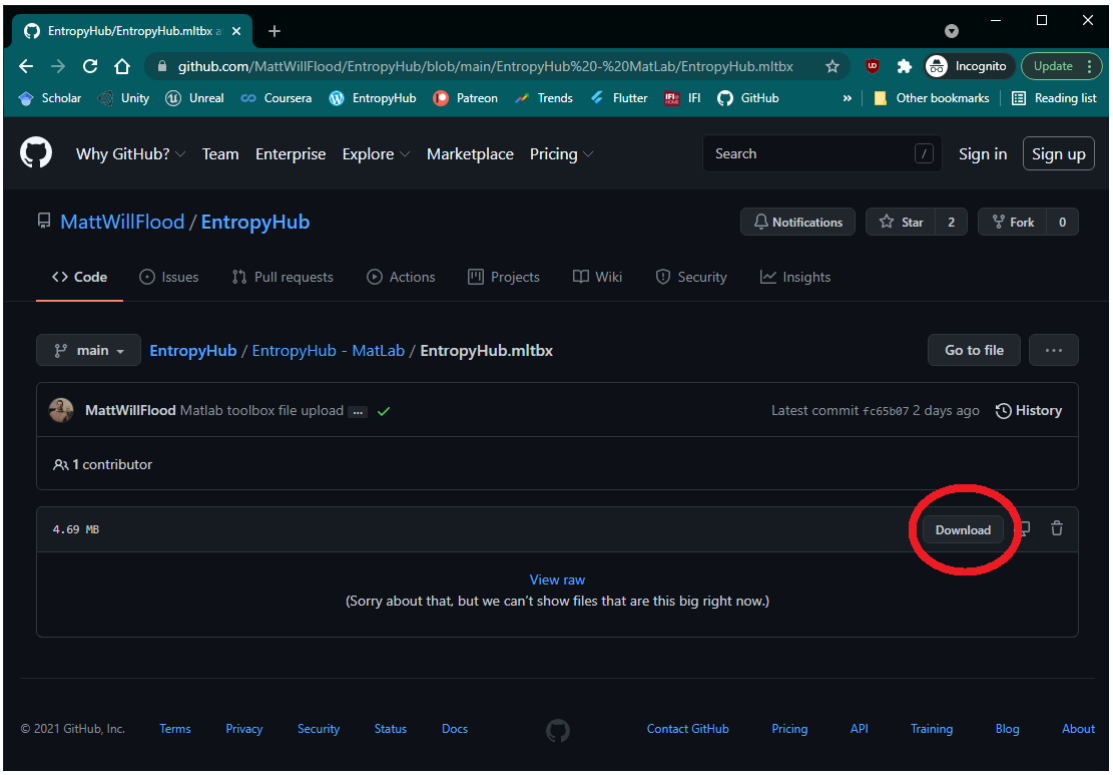
Method 3.

1. Go to the MatLab folder in the EntropyHub Github repository.

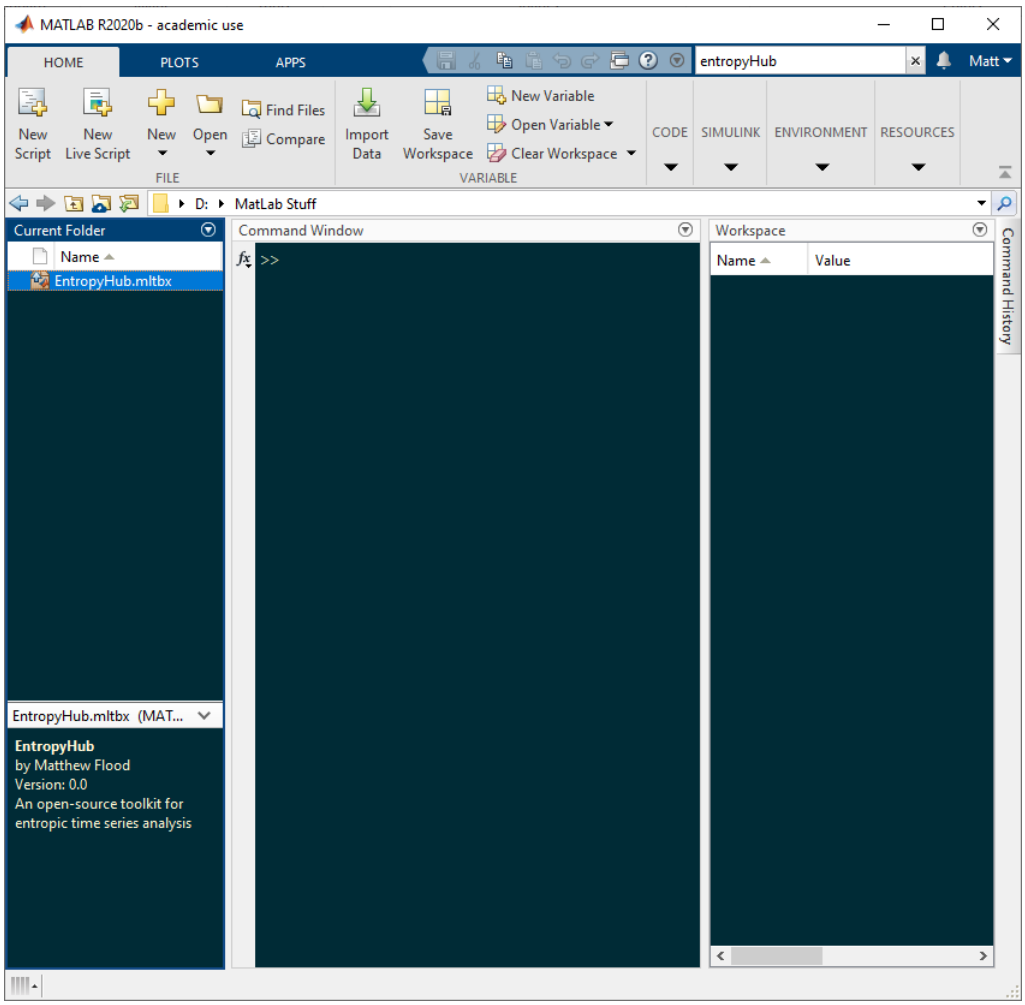


2. Open on the link to the toolbox file (EntropyHub.mltbx) and click the button labelled 'Download'.





3. In MatLab, navigate the current folder to the directory where the EntropyHub.mltbx file is saved. Open the file and click install.


MATLAB R2020b - academic use
HOME
PLOTS
APPS
entropyHub
Matt
New Script
New Live Script
New
Open
Find Files
Compare
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
CODE
SIMULINK
ENVIRONMENT
RESOURCES
FILE
VARIABLE
D: MatLab Stuff
Current Folder
Name
EntropyHub.mltbx
Command Window
>>
Workspace
Name
Value
Command History
EntropyHub.mltbx (MAT...
EntropyHub
by Matthew Flood
Version: 0.0
An open-source toolkit for
entropic time series analysis


Install EntropyHub
EntropyHub by Matthew Flood
Updated on 12 May 2021
Version: 0.0
An open-source toolkit for entropic time series analysis
Install
Cancel

2.2 Python

System Requirements

There are several package dependencies which will be installed alongside EntropyHub:

NumPy

SciPy

Matplotlib

PyEMD

Requests

EntropyHub was designed using Python 3 and thus is not intended for use with Python 2. Python versions >= 3.6 are required for using EntropyHub.

There are 2 ways to install EntropyHub for Python. Method 1 is **strongly recommended**.

Method 1.

1. Python comes with an inbuilt package management system, `pip`. Pip can install, update, or delete any official package. You can install packages via the command line by entering:

   ```
   >>> pip install EntropyHub
   ```

   If using a Python IDE, it is recommended to restart the terminal after installation.

2. To use EntropyHub, import the module with the following command:

   ```
   >>> import EntropyHub
   ```

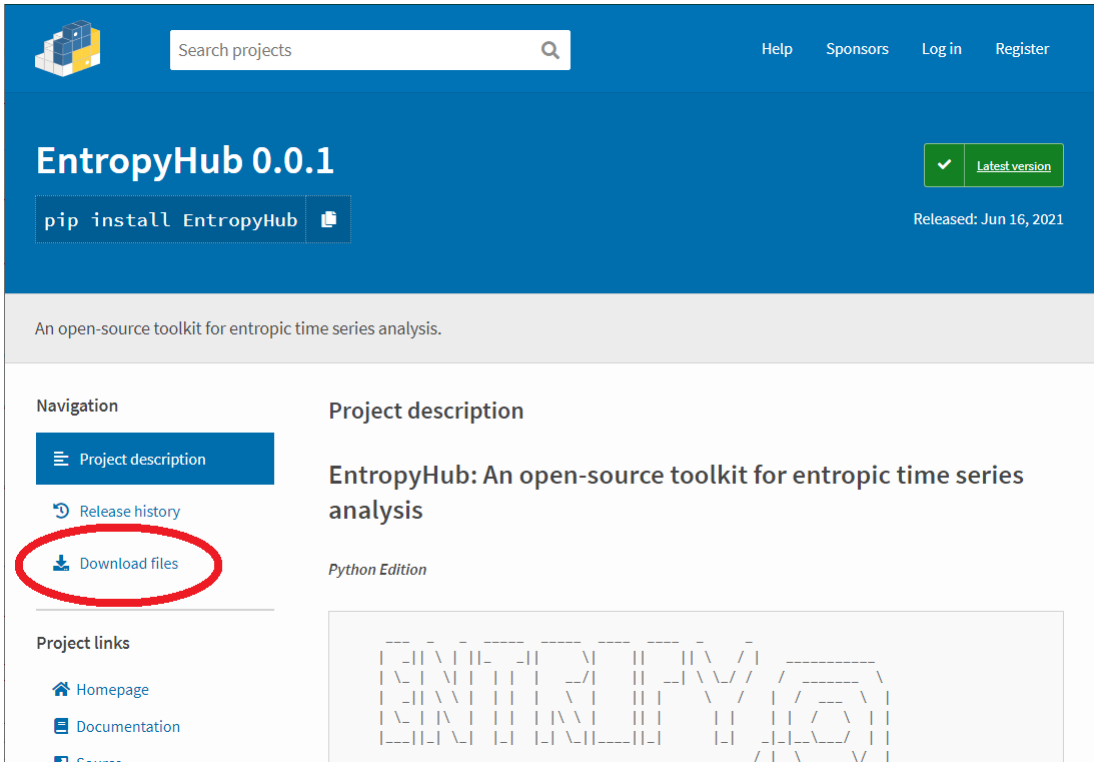
   or in abbreviated form,

   ```
   >>> import EntropyHub as EH
   ```

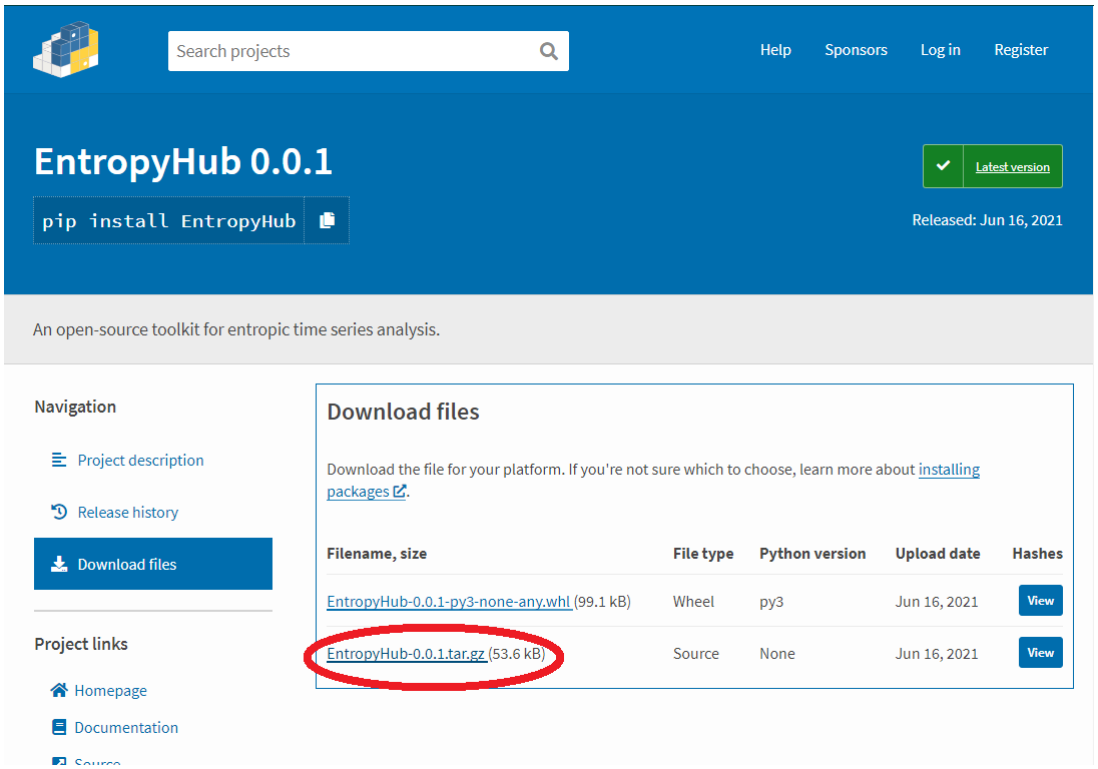
Method 2.

***Note:** installation with Method 2 requires the latest version of **`wheel`** to be previously installed in Python.

1. Visit the EntropyHub PyPI page (or the EntropyHub GitHub page).



2. Download the latest .tar.gz folder *(EntropyHub.x.x.x.tar.gz)* from the 'download files' button on the left-hand side.



3. Extract the files into a local directory.

4. Open a command prompt or terminal window and navigate to the root directory where **setup.py** is located.

5. In the command line, enter:

```
>>> python setup.py install
```

   *Ensure that an up-to-date version of `setuptools` is installed:

```
>>> python -m pip install --upgrade setuptools
```

6. To use EntropyHub, import the module with the following command:

```
>>> import EntropyHub
```

   or in abbreviated form,

```
>>> import EntropyHub as EH
```

2.3 Julia

There are several package dependencies which will be installed alongside EntropyHub:

DataInterpolations.jl

StatsFuns.jl

StatsBase.jl

GroupSlices.jl

Clustering.jl

Combinatorics.jl

DSP.jl

FFTW.jl

HTTP.jl

Statistics.jl

DelimitedFiles.jl

Random.jl

LinearAlgebra.jl

Plots.jl

There are 2 ways to install EntropyHub in Julia. Method 1 is recommended.

Method 1.

1. In your Julia IDE, open the package REPL and enter:

```
julia> ]
pkg> add EntropyHub
```

   or alternatively:

```
using Pkg
Pkg.add("EntropyHub")
```

2. To use EntropyHub in Julia, enter:

```
using EntropyHub
```

   or import specific functions:

```
using EntropyHub: SampEn, MSobject, MSEn
```

Method 2.

1. Open the Julia package REPL and enter the following:

```
julia> ]
pkg> add https://github.com/MattWillFlood/EntropyHub.jl
```

2. To use EntropyHub in Julia, enter:

```
using EntropyHub
```

or import specific functions:

```
using EntropyHub: SampEn, MSobject, MSEn
```

3 Functions

Sections 3.1 – 3.8 outline the command line syntax of each function with descriptions of every argument and returned value, as well as references to the source literature. The order of the function commands under the syntax subheading is MatLab first, Python second, Julia third.

NOTE

For concision, function commands written in the following sections using **Python** syntax exclude the module prefix which would otherwise be required, i.e. `EntropyHub.SampEn()` is written as `SampEn()`.

NOTE

Python functions in EntropyHub are based primarily on the `Numpy` module. Arguments in python functions with the **`np.`** prefix refer to `numpy` functions.

3.1 Base Entropy Functions

3.1.1 ApEn: Approximate Entropy

Syntax

```
[Ap, Phi] = ApEn(Sig, 'm', 2, 'tau', 1, 'r', 0.2*std(Sig), 'Logx', exp(1))
Ap, Phi = ApEn(Sig, m = 2, tau = 1, r = 0.2*np.std(Sig), Logx = np.exp(1))
Ap, Phi = ApEn(Sig, m = 2, tau = 1, r = 0.2*std(Sig), Logx = exp(1))
```

Arguments

Sig	Time series signal, a vector of length > 10.
m	Embedding dimension, a positive integer.
tau	Time delay, a positive integer.
r	Distance threshold, a positive scalar.
Logx	Logarithm base in the entropy formula, a positive scalar.

Outputs

Ap	Approximate entropy estimates, a vector of length m+1. **The first value of **Ap** is the zeroth estimate, i.e. $-\Phi_1$, and the last value of **Ap** is the estimate for the specified **m**.
Phi	The number of matched state vectors for each embedding dimension from 0 to m+1.

References [1]

3.1.2 SampEn: Sample Entropy

Syntax

```
[Samp, A, B, Vcp_Ka_Kb] = SampEn(Sig, 'm', 2, 'tau', 1, 'r', 0.2*std(Sig), 'Logx', exp(1), 'Vcp', false)
Samp, A, B, Vcp_Ka_Kb = SampEn(Sig, m = 2, tau = 1, r = 0.2*np.std(Sig), Logx = np.exp(1), Vcp = False)
Samp, A, B, Vcp_Ka_Kb = SampEn(Sig, m = 2, tau = 1, r = 0.2*std(Sig), Logx = exp(1), Vcp = false)
```

Arguments

Sig	Time series signal, a vector of length > 10.
m	Embedding dimension, a positive integer.
tau	Time delay, a positive integer.
r	Distance threshold, a positive scalar.
Logx	Logarithm base in the entropy formula, a positive scalar.
Vcp	Option to return the variance of the conditional probabilities and the number of overlapping matching vector pairs of lengths [3]

Outputs

Samp	Sample entropy estimates, a vector of length **m+1**. **The first value of **Samp** is the zeroth estimate, i.e. $\frac{1}{2}Log(N(N-1))$–$Log(A_1)$, and the last value of **Samp** is the estimate for the specified **m**.
A	The number of matched state vectors for each embedding dimension from 0 to m.
B	The number of matched state vectors for each embedding dimension from 1 to m+1.
Vcp_Ka_Kb	A vector/tuple of three values corresponding to: **Vcp_Ka_Kb(1) = V_{cp}** Estimate of the variance of conditional probabilities used in the estimation of the sample entropy estimate. **Vcp_Ka_Kb(2) = K_a** The number of overlapping vector pairs of length m+1 **Vcp_Ka_Kb(3) = K_b** The number of overlapping vector pairs of length m

Note: Vcp is undefined for the zeroth embedding dimension (m = 0) and due to the computational demand, **will take substantially more time to return function outputs. See Appendix B in [3] for more info.

References [2] [3]

3.1.3 FuzzEn: Fuzzy Entropy

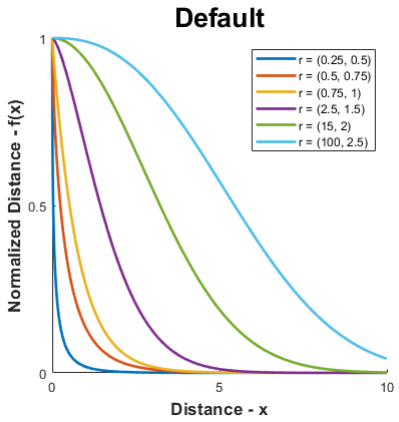
Syntax

```
[Fuzz, Ps1, Ps2] = FuzzEn(Sig, 'm', 2, 'tau', 1, 'Fx', 'default', 'r', [0.2, 2], 'Logx', exp(1))
Fuzz, Ps1, Ps2 = FuzzEn(Sig, m = 2, tau = 1, Fx = "default", r = (0.2, 2), Logx = np.exp(1))
Fuzz, Ps1, Ps2 = FuzzEn(Sig, m = 2, tau = 1, Fx = "default", r = (0.2, 2), Logx = exp(1))
```

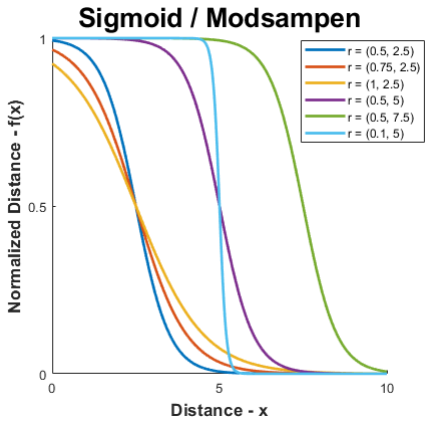
Arguments

Sig Time series signal, a vector of length > 10.
m Embedding dimension, a positive integer.
tau Time delay, a positive integer.
Fx Type of fuzzy function for distance transformation, one of the following strings:

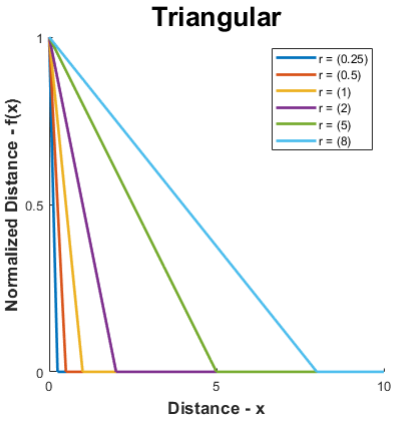
"default" $f(x) = exp(-\frac{x^{r_2}}{r_1})$



"sigmoid"/"modsampen" $f(x) = (1 + exp(\frac{x-r_2}{r_1}))^{-1}$

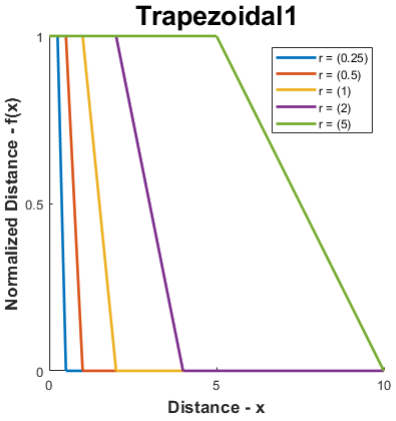


"triangular" $f(x) = \begin{cases} 1 - \frac{x}{r}, & x \leq r \\ 0, & x > r \end{cases}$



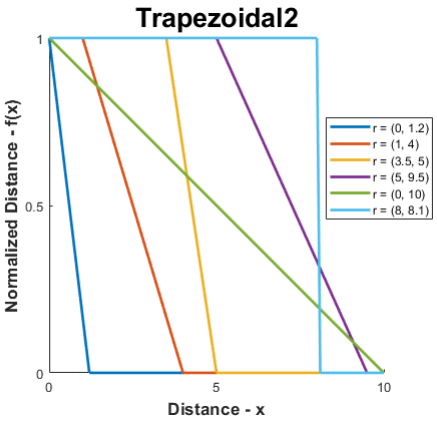
"trapezoidal1"

$$f(x) = \begin{cases} 1, & x < r \\ 2 - \frac{x}{r}, & r \leq x < 2r \\ 0, & x \geq 2r \end{cases}$$



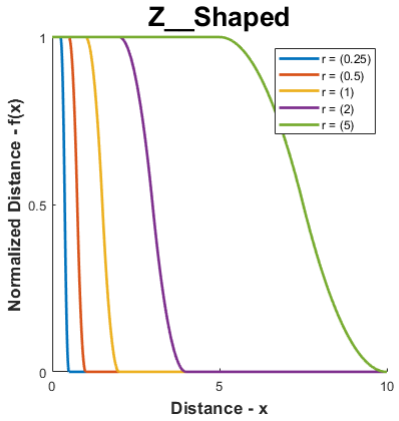
"trapezoidal2"

$$f(x) = \begin{cases} 1, & x < r \\ 1 - \frac{x - r_{min}}{r_{max} - r_{min}}, & r \leq x < r_{max} \\ 0, & x \geq r_{max} \end{cases}$$



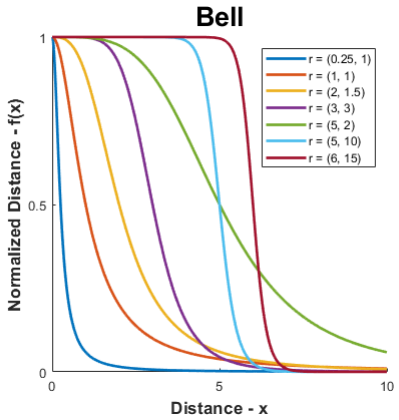
"z_shaped"

$$f(x) = \begin{cases} 1, & x < r \\ 1 - 2\left(\frac{x-r}{r}\right)^2, & r < x < \frac{3}{2}r \\ 2\left(\frac{x-r}{r}\right)^2, & \frac{3}{2}r < x < 2r \\ 0, & x > 2r \end{cases}$$



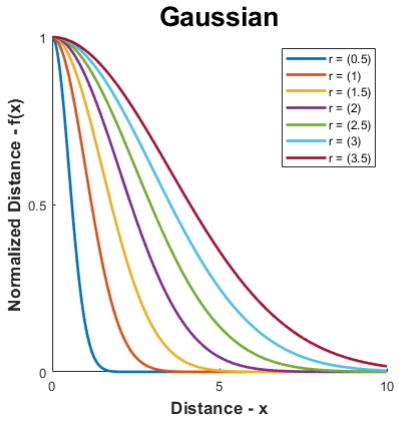
"bell"

$$f(x) = \frac{1}{1+\left|\frac{x}{r_1}\right|^{r_2}}$$



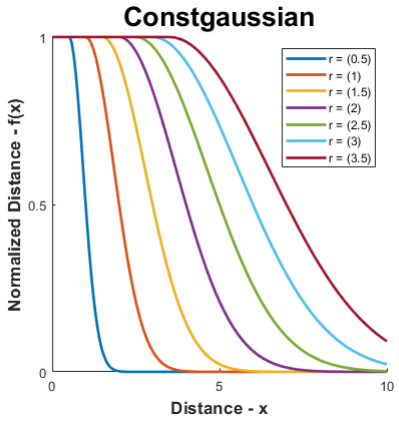
"gaussian"

$$f(x) = exp(-\frac{x^2}{2r^2})$$



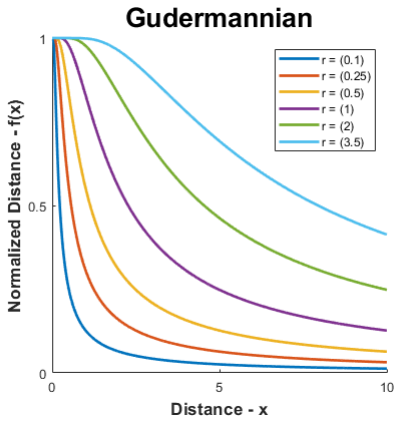
"constgaussian"

$$f(x) = \begin{cases} 1, & x < r_{min} \\ exp(-\log(2)\frac{(x-r)^2}{r}), & x \geq r_{max} \end{cases}$$



"gudermannian"

$g(x) = atan(\frac{tanh(r_1)}{x})$
$f(x) = \frac{g(x)}{g(x_{max})}$
Note: Distances are normalized w.r.t. maximum distance relative to each state vector.



r Parameters of the fuzzy function specified by **Fx**, a 1 element scalar or a 2 element tuple of positive values depending on the fuzzy function as shown above.

Default	Two element tuple (or vector in MatLab)
Sigmoid/ModSampEn	Two element tuple (or vector in MatLab)
Trapezoidal2	Two element tuple (or vector in MatLab)
Bell	Two element tuple (or vector in MatLab)
Trapezoidal1	A scalar value
Triangular	A scalar value
Z_Shaped	A scalar value
Gaussian	A scalar value
ConstGaussian	A scalar value
Gudermannian	A scalar value

Logx Logarithm base in the entropy formula, a positive scalar.

Outputs

Fuzz Fuzzy entropy estimates for each embedding dimension 1:m.
Ps1 The average fuzzy distances for embedding dimensions 1:m.
Ps2 The average fuzzy distances for embedding dimensions 2:m+1.

References

[4] [5] [6]

3.1.4 K2En: Kolmogorov Entropy

Syntax

```
[K2, Ci] = K2En(Sig, 'm', 2, 'tau', 1, 'r', 0.2*std(Sig), 'Logx', exp(1))
K2, Ci = K2En(Sig, m = 2, tau = 1, r = 0.2*np.std(Sig), Logx = np.exp(1))
K2, Ci = K2En(Sig, m = 2, tau = 1, r = 0.2*std(Sig), Logx = exp(1))
```

Arguments

Sig	Time series signal, a vector of length > 10.
m	Embedding dimension, an integer.
tau	Time delay, a positive integer.
r	Distance threshold, a positive scalar.
Logx	Logarithm base in the entropy formula, a positive scalar.

Outputs

K2	Kolmogorov entropy estimates for each embedding dimension from 1 to m
Ci	The correlation sum for each embedding dimension from 1 to m.

References [7] [8]

3.1.5 PermEn: Permutation Entropy

Syntax

```
[Perm, Pnorm, cPE] = PermEn(Sig, 'm', 2, 'tau', 1, 'Typex', 'none', 'tpx', [], 'Logx', 2, 'Norm', false)
Perm, Pnorm, cPE = PermEn(Sig, m = 2, tau = 1, Typex = 'none', tpx = -1, Logx = 2, Norm = False)
Perm, Pnorm, cPE = PermEn(Sig, m = 2, tau = 1, Typex = "none", tpx = nothing, Logx = 2, Norm = false)
```

Arguments

Sig Time series signal, a vector of length > 10.
It is recommended that length of Sig (N) > 5m!
[*Amigoetal.*, *Europhys.Lett.*83 : 60005, 2008]

m Embedding dimension, an integer > 1.

tau Time delay, a positive integer.

Typex Variant of permutation entropy, one of the following strings:

"finegrain"	Fine-grained permutation entropy [10]
"modified"	Modified permutation entropy [11]
"weighted"	Weighted permutation entropy [12]
"ampaware"	Amplitude-aware permutation entropy [13]
"edge"	Edge permutation entropy [14]
"uniquant"	Uniform quantization-based permutation entropy [15]
"phase"	Phase permutation entropy [17]

tpx Tuning parameter for the permutation entropy specified by the **Typex** argument.

finegrain	**tpx** is the α parameter, a positive scalar (default: 1)
ampaware	**tpx** is the A parameter, a value in range [0 1] (default: 0.5)
edge	**tpx** is the r sensitivity parameter, a scalar > 0 (default: 1)
uniquant	**tpx** is the L parameter, an integer > 1 (default: 4).
phase	**tpx** is the option to unwrap the phase shift of the Hilbert-transformed data sequence, either [ ] or 1 (default: [ ]).

Logx Logarithm base in the entropy formula, a positive scalar.

Norm Normalisation of **Pnorm** value, a boolean operator:
`false` normalises w.r.t *Log*(# of permutation symbols [m]) - default
`true` normalises w.r.t *Log*(# of all possible permutations [m!])
* Note: Normalised permutation entropy is undefined for `m = 1`.
** Note: When `Typex = uniquant` and `Norm = true`, normalisation of `Perm` is calculated w.r.t. $Log(tpx^m)$

Outputs

Perm Permutation entropy estimates for embedding dimensions 1:m.

Pnorm Normalised Permutation entropy estimates.

cPE Conditional permutation entropy [16]

References

[9] [10] [11] [12] [13] [14] [15] [16] [17]

3.1.6 CondEn: *corrected* Conditional Entropy

Syntax

```
[Cond, SEw, SEz] = CondEn(Sig, 'm', 2, 'tau', 1, 'c', 6, 'Logx', exp(1), 'Norm', false)
Cond, SEw, SEz = CondEn(Sig, m = 2, tau = 1, c = 6, Logx = np.exp(1), Norm = False)
Cond, SEw, SEz = CondEn(Sig, m = 2, tau = 1, c = 6, Logx = exp(1), Norm = false)
```

Arguments

Sig	Time series signal, a vector of length > 10.
m	Embedding dimension, an integer > 1.
tau	Time delay, a positive integer.
c	Number of symbols in symbolic transformation, in integer > 1
Logx	Logarithm base in the entropy formula, a positive scalar.
Norm	Normalization of **Cond** value: `true` no normalisation (default) `false` normalises w.r.t Shannon entropy of data sequence **Sig**

Outputs

Cond	Corrected conditional entropy estimate
SEw	Shannon entropy estimate for **m**.
SEz	Shannon entropy estimate for **m+1**.

References [18]

3.1.7 DistEn: Distribution Entropy

Syntax

```
[Dist, Ppi] = DistEn(Sig, 'm', 2, 'tau', 1, 'Bins', 'sturges', 'Logx', 2, 'Norm', true)
Dist, Ppi = DistEn(Sig, m = 2, tau = 1, Bins = 'sturges', Logx = 2, Norm = True)
Dist, Ppi = DistEn(Sig, m = 2, tau = 1, Bins = "sturges", Logx = 2, Norm = true)
```

Arguments

Sig	Time series signal, a vector of length > 10.
m	Embedding dimension, a positive integer.
tau	Time delay, a positive integer.
Bins	Histogram bin selection method, in integer > 1 indicating the number of bins, or one of the following strings: "sturges", "sqrt", "rice", "doanes" [default: "sturges"] ¿¿ More info on binning methods.
Logx	Logarithm base in the entropy formula, a positive scalar. (Enter 0 for natural logarithm)
Norm	Normalization of **Dist** value: `false` no normalisation `true` normalises w.r.t number of histogram bins (default)

Outputs

Dist	Distribution entropy estimate.
Phi	Probability of each histogram bin.

References [19]

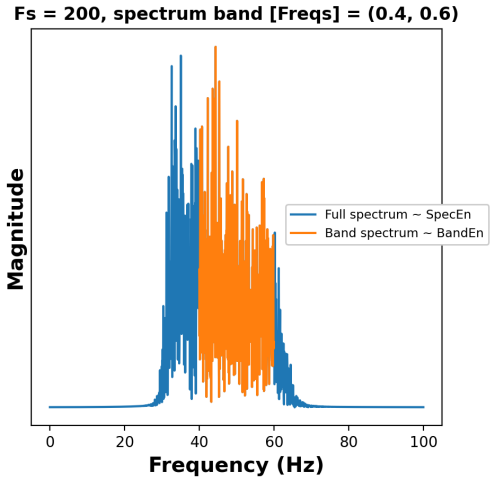
3.1.8 SpecEn: Spectral Entropy

Syntax

```
[Spec, BandEn] = SpecEn(Sig, 'N', 2*length(Sig)+1, 'Freqs', [0,1], 'Logx', exp(1), 'Norm', true)
Spec, BandEn = SpecEn(Sig, N = 2*len(Sig) + 1, Freqs = (0,1), Logx = np.exp(1), Norm = True)
Spec, BandEn = SpecEn(Sig, N = 2*length(Sig) + 1, Freqs = (0,1), Logx = exp(1), Norm = true)
```

Arguments

Sig Time series signal, a vector of length > 10.
N Resolution of the N-point fft, an integer > 1.
Freqs Normalised band edge-frequencies for calculating the band entropy (BandEn), a 2 element tuple with values in range [0,1] where 1 is the Nyquist frequency.
* When no edge frequencies are provided, BandEn==SpecEn



Logx Logarithm base in the entropy formula, a positive scalar.
Norm Normalization of **Spec** value:
`false` no normalisation
`true` normalises **Spec** w.r.t number of Nyquist frequency value, and **BandEn** w.r.t. range of frequencies in the band given by Freqs. (default)

Outputs

Spec Spectral entropy estimate.
BandEn Spectral band entropy estimate.

References

[20] [21]

Note: In contrast to other *Base* entropies, spectral entropy (SpecEn) is not derived from information theory or dynamical systems theory, and instead is an estimate of the frequency spectrum curve estimated using the discrete time Fourier transform.

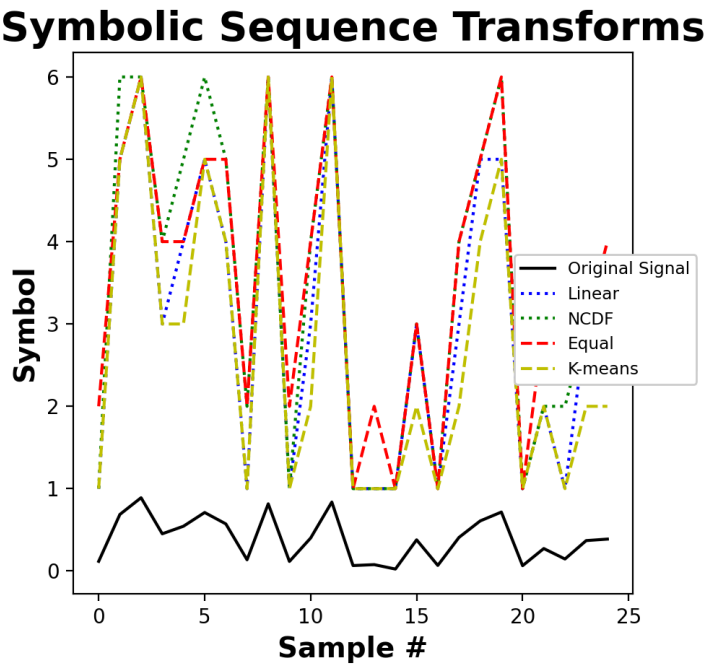
3.1.9 DispEn: Dispersion Entropy

Syntax

```
[Dispx, RDE] = DispEn(Sig, 'm', 2, 'tau', 1, 'c', 3, 'Typex', 'ncdf', 'Logx', exp(1), 'Fluct', false, 'Norm', false, 'rho', 1)
Dispx, RDE = DispEn(Sig, m = 2, tau = 1, c = 3, Typex = 'ncdf', Logx = exp(1), Fluct = False, Norm = False, rho = 1)
Dispx, RDE = DispEn(Sig, m = 2, tau = 1, c = 3, Typex = "ncdf", Logx = exp(1), Fluct = false, Norm = false, rho = 1)
```

Arguments

Sig Time series signal, a vector of length > 10.
m Embedding dimension, a positive integer.
tau Time delay, a positive integer.
c Number of symbols in transform, an integer > 1.
Typex Type of symbolic sequence transform, one of the following strings:
- **"ncdf"** Normalised cumulative distribution function [22]
- **"kmeans"** K-means clustering algorithm. **Note: The *"kmeans"* algorithm uses random initialization conditions. This causes results to vary slightly each time it is called.
- **"linear"** Linear segmentation of signal range
- **"finesort"** Fine-sorted dispersion entropy [25]
- **"equal"** Approx. equal number of symbols.



Logx Logarithm base in the entropy formula, a positive scalar.
Fluct When **true**, returns the fluctuation-based dispersion entropy [23]
Norm Normalisation of **Dispx** and **RDE** values, a boolean operator:
- `false` no normalisation
- `true` normalises w.r.t number of possible dispersion patterns (default)

rho *If `Typex = "finesort"`, **rho** is the tuning parameter, a positive scalar (default: 1)

Outputs

Dispx	Dispersion entropy estimate.
RDE	Reverse dispersion entropy estimate. [24]

References [22] [23] [24] [25]

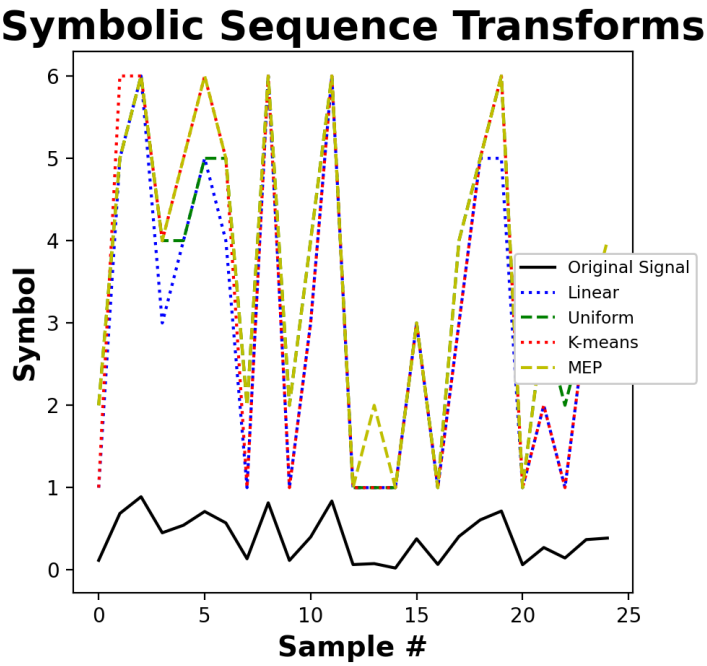
3.1.10 SyDyEn: Symbolic Dynamic Entropy

Syntax

```
[SyDy, Zt] = SyDyEn(Sig, 'm', 2, 'tau', 1, 'c', 3, 'Typex', 'MEP', 'Logx', exp(1), 'Norm', true)
SyDy, Zt = SyDyEn(Sig, m = 2, tau = 1, c = 3, Typex = 'MEP', Logx = np.exp(1), Norm = True)
SyDy, Zt = SyDyEn(Sig, m = 2, tau = 1, c = 3, Typex = "MEP", Logx = exp(1), Norm = true)
```

Arguments

Sig Time series signal, a vector of length > 10.
m Embedding dimension, a positive integer.
tau Time delay, a positive integer.
c Number of symbols, an integer > 1.
Typex Type of symbolic sequence partitioning, one of the following strings:
- **"MEP"** Maximum entropy partitioning [27]
- **"kmeans"** K-means clustering algorithm. *Note: The "kmeans" algorithm uses random initialization conditions. This causes results to vary slightly when repeatedly called.
- **"linear"** Linear segmentation of signal range
- **"uniform"** Approx. equal number of symbols.



Logx Logarithm base in the entropy formula, a positive scalar.
Norm Normalisation of **SyDy** value, a boolean operator:
- `false` no normalisation
- `true` normalises w.r.t number of possible vector permutations (c^{m+1})

Outputs

SyDy	Symbolic Dynamic entropy estimate.
Zt	Symbolic sequence of transformed time series.

References [26] [27] [28]

3.1.11 IncrEn: Increment Entropy

Syntax

```
Incr = IncrEn(Sig, 'm', 2, 'tau', 1, 'R', 4, 'Logx', exp(1), 'Norm', false)
Incr = IncrEn(Sig, m = 2, tau = 1, R = 4, Logx = np.exp(1), Norm = False)
Incr = IncrEn(Sig, m = 2, tau = 1, R = 4, Logx = exp(1), Norm = false)
```

Arguments

Sig	Time series signal, a vector of length > 10.
m	Embedding dimension, an integer > 1.
tau	Time delay, a positive integer.
R	Quantifying resolution, a positive scalar.
Logx	Logarithm base in the entropy formula, a positive scalar.
Norm	Normalisation of **Incr** value. `false` no normalisation (default) `true` normalises w.r.t. embedding dimension (**m-1**)

Outputs

Incr	Increment entropy estimate.

References [29] [30] [31]

3.1.12 CoSiEn: Cosine Similarity Entropy

Syntax

```
[CoSi, Bm] = CoSiEn(Sig, 'm', 2, 'tau', 1, 'r', 0.1, 'Logx', 2, 'Norm', 0)
CoSi, Bm = CoSiEn(Sig, m = 2, tau = 1, r = 0.1, Logx = 2, Norm = 0)
CoSi, Bm = CoSiEn(Sig, m = 2, tau = 1, r = 0.1, Logx = 2, Norm = 0)
```

Arguments

Sig	Time series signal, a vector of length > 10.
m	Embedding dimension, an integer > 1.
tau	Time delay, a positive integer.
r	Angular threshold, a value in range $[0 < \mathbf{r} < 1]$
Logx	Logarithm base in the entropy formula, a positive scalar.
Norm	Normalisation of **Sig**, an integer in range [0 4]: 0 - no normalisation (default) 1 - median removed 2 - mean removed 3 - normalised to unit variance 4 - normalised to range [-1 1]

Outputs

CoSi	Cosine similarity entropy estimate.
Bm	Sum of global probabilities.

References [32]

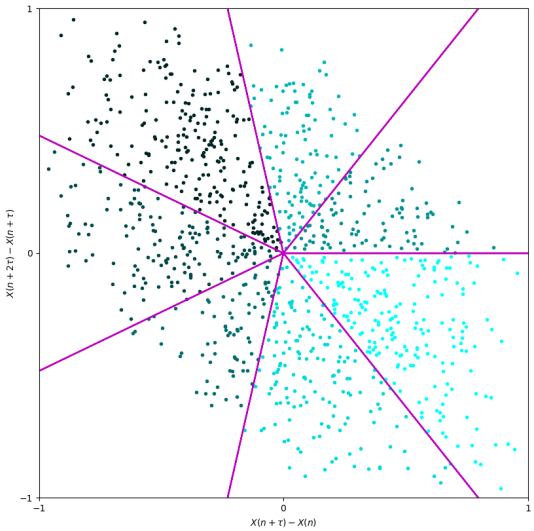
3.1.13 PhasEn: Phase Entropy

Syntax

```
Phas = PhasEn(Sig, 'K', 4, 'tau', 1, 'Logx', exp(1), 'Norm', true, 'Plotx', false)
Phas = PhasEn(Sig, K = 4, tau = 1, Logx = np.exp(1), Norm = True, Plotx = False)
Phas = PhasEn(Sig, K = 4, tau = 1, Logx = exp(1), Norm = true, Plotx = false)
```

Arguments

Sig	Time series signal, a vector of length > 10.
K	Number of angular partitions, an integer > 1. **Note:** Angular partitions of the second-order difference plot (SODP) are first split between 0 and n degrees w.r.t. the positive x-axis. As this point is somewhat arbitrary, it is recommended to use even-numbered (preferably multiples of 4) partitions for sake of symmetry.
tau	Time delay, a positive integer.
Logx	Logarithm base in the entropy formula, a positive scalar.
Norm	Normalisation of **Phas**: `false` no normalisation (default) `true` normalises w.r.t. the number of partitions **Log(K)**
Plotx	When `Plotx == true`, returns SODP (default: `false`) The example below depicts the SODP of normally distributed random numbers with 7 angular partitions (**K**).



Outputs

Phas	Phase entropy estimate.

References

[33]

3.1.14 SlopEn: Slope Entropy

Syntax

```
Slop = SlopEn(Sig, 'm', 2, 'tau', 1, 'Logx', 2, 'Lvls', [5, 45], 'Norm', true)
Slop = SlopEn(Sig, m = 2, tau = 1, Logx = 2, Lvls = (5, 45), Norm = True)
Slop = SlopEn(Sig, m = 2, tau = 1, Logx = 2, Lvls = [5, 45], Norm = true)
```

Arguments

Sig	Time series signal, a vector of length > 10.
m	Embedding dimension, an integer > 1.
tau	Time delay, a positive integer.
Logx	Logarithm base in the entropy formula, a positive scalar. Enter 0 for natural logarithm.
Lvls	Angular thresolds, a vector (or tuple in python) of monotonically increasing values in the range [0 90] degrees
Norm	Normalisation of **Slop**: `false` no normalisation `true` normalises w.r.t. the number of unique patterns found.

Outputs

Slop	Slope entropy estimates, a vector of length m-1 where values correspond to embedding dimensions [2, ..., m]

References [34]

3.1.15 BubbEn: Bubble Entropy

Syntax

```
[Bubb, H] = BubbEn(Sig, 'm', 2, 'tau', 1, 'Logx', exp(1))
Bubb, H = BubbEn(Sig, m = 2, tau = 1, Logx = np.exp(1))
Bubb, H = BubbEn(Sig, m = 2, tau = 1, Logx = exp(1))
```

Arguments

Sig	Time series signal, a vector of length > 10.
m	Embedding dimension, an integer > 1.
tau	Time delay, a positive integer.
Logx	Logarithm base in the entropy formula, a positive scalar.

Outputs

Bubb	Bubb entropy estimate.
H	Conditional Rényi entropy

References [35]

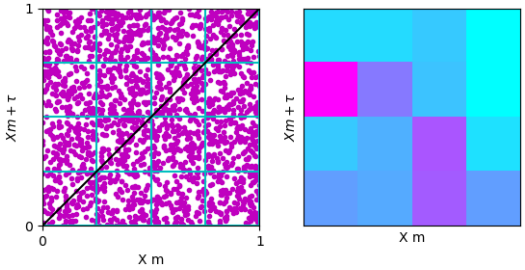
3.1.16 GridEn: Gridded Distribution Entropy

Syntax

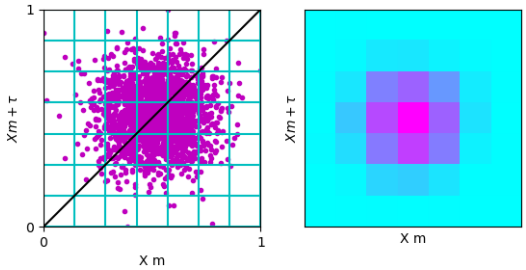
```
[GDE, GDR, PIx, GIx, SIx, AIx] = GridEn(Sig, 'm', 3, 'tau', 1, 'Logx', exp(1), 'Plotx', false)
GDE, GDR, PIx, GIx, SIx, AIx = GridEn(Sig, m = 3, tau = 1, Logx = np.exp(1), Plotx = False)
GDE, GDR, PIx, GIx, SIx, AIx = GridEn(Sig, m = 3, tau = 1, Logx = exp(1), Plotx = false)
```

Arguments

Sig Time series signal, a vector of length > 10.
m Number of grid divisions, an integer > 1.
tau Time delay, a positive integer.
Logx Logarithm base in the entropy formula, a positive scalar.
Plotx When `Plotx == true`, returns Poicaré plot and a bivariate histogram of the grid point distribution (default: `false`)



Poincaré plot and bivariate histogram of uniform random number sequence ($m = 4$).



Poincaré plot and bivariate histogram of white noise ($m = 7$).

Outputs

GDE Gridded distribution entropy estimate.
GDR Gridded distribution rate.
PIx Percentage of points below the line of identity (LI). [38]
GIx Proportion of point distances above the LI. [40]
SIx Ratio of phase angles (w.r.t. LI) of the points above the LI.[39]
AIx Ratio of the cumulative area of sectors of points above the LI.[37]

References

[36] [37] [38] [39] [40]

3.1.17 EnofEn: Entropy of Entropy

Syntax

```
[EoE, AvEn, S2] = EnofEn(Sig, 'tau', 10, 'S', 10, 'Xrange', [min(Sig) max(Sig)],
'Logx', exp(1))
EoE, AvEn, S2 = EnofEn(Sig, tau = 10, S = 10, Xrange = (np.min(Sig), np.max(Sig)),
Logx = np.exp(1))
EoE, AvEn, S2 = EnofEn(Sig, tau = 10, S = 10, Xrange = (min(Sig),max(Sig)), Logx
= exp(1)
```

Arguments

Sig	Time series signal, a vector of length > 10.
tau	Window length, an integer > 1 and < length(Sig)
S	Number of slices (s1), an integer > 1
Xrange	The min and max of the range included in the division of slices, a two-element tuple where Xrange[0] <= Xrange[1]
Logx	Logarithm base in the entropy formula, a positive scalar.

Outputs

EoE	Entropy of entropy estimate.
AvEn	Average Shannon entropy across all windows.
S2	Number of levels (S2) used for the given tau and S.

References [41]

3.1.18 AttnEn: Attention Entropy

Syntax

```
[Attn, Hxx, Hnn, Hxn, Hnx] = EnofEn(Sig, 'Logx', 2)
Attn, Hxx, Hnn, Hxn, Hnx = EnofEn(Sig, Logx = 2)
Attn, Hxx, Hnn, Hxn, Hnx = EnofEn(Sig, Logx = 2)
```

Arguments

Sig	Time series signal, a vector of length > 10.
Logx	Logarithm base in the entropy formula, a positive scalar. Enter 0 for natural logarithm.

Outputs

Attn	Attention entropy estimate.
Hxx	Entropy of local-maxima intervals
Hnn	Entropy of local-minima intervals
Hxn	Entropy of intervals between local maxima and subsequent minima
Hnx	Entropy of intervals between local minima and subsequent maxima

References [42]

3.1.19 DivEn: Diversity Entropy

Syntax

```
[Div, CDEn, Bm] = DivEn(Sig, 'm', 2, 'tau', 1, 'r', 5, 'Logx', exp(1))
Div, CDEn, Bm = DivEn(Sig, m = 2, tau = 1, r = 5, Logx = np.exp(1))
Div, CDEn, Bm = DivEn(Sig, m = 2, tau = 1, r = 5, Logx = exp(1))
```

Arguments

Sig	Time series signal, a vector of length > 10.
m	Embedding dimension, an integer > 1.
tau	Time delay, a positive integer.
r	Histogram bins, either: - an integer > 1 representing the number of bins - a vector array of 3 or more increasing values in range [-1 1] representing the bin edges including the rightmost edge.
Logx	Logarithm base in the entropy formula, a positive scalar. Enter 0 for natural logarithm.

Outputs

Div	Diversity entropy estimate.
CDEn	Cumulative diversity entropy estimate.
Bm	Probability from each histogram bin.

References [43] [44]

3.1.20 RangEn: Range Entropy

Syntax

```
[Rangx, A, B] = RangEn(Sig, 'm', 2, 'tau', 1, 'r', 0.2, 'Methodx', 'SampEn', 'Logx', exp(1))
Rangx, A, B = RangEn(Sig, m = 2, tau = 1, r = 0.2, Methodx = "SampEn", Logx = np.exp(1))
Rangx, A, B = RangEn(Sig, m = 2, tau = 1, r = 0.2, Methodx = "SampEn", Logx = exp(1))
```

Arguments

Sig	Time series signal, a vector of length > 10.
m	Embedding dimension, an integer > 1.
tau	Time delay, a positive integer.
r	Distance threshold, a positive scalar
Methodx	Method for entropy estimation, either "ApEn" or "SampEn"
Logx	Logarithm base in the entropy formula, a positive scalar. Enter 0 for natural logarithm.

Outputs

Rangx	Range entropy estimate.
A	Number of matched state vectors of length m+1.
B	Number of matched state vectors of length m.

References [45] [1] [2]

3.2 Cross-Entropy Functions

3.2.1 XApEn: Cross-Approximate Entropy

Syntax

```
[XAp, Phi] = XApEn(Sig1, Sig2, 'm', 2, 'tau', 1, 'r', 0.2*std_pooled(Sig1, Sig2), 'Logx', exp(1))
XAp, Phi = XApEn(Sig1, Sig2, m = 2, tau = 1, r = 0.2*std_pooled(Sig1, Sig2), Logx = np.exp(1))
XAp, Phi = XApEn(Sig1, Sig2, m = 2, tau = 1, r = 0.2*std_pooled(Sig1, Sig2), Logx = exp(1))
```

Arguments

Sig1	First data sequence, a vector of > 10 elements.
Sig2	Second data sequence, a vector of > 10 elements. **NOTE: XApEn is direction-dependent**. Thus, **Sig1** is used as the template data sequence, and the **Sig2** is the matching sequence.
m	Embedding dimension, a positive integer.
tau	Time delay, a positive integer.
r	Distance threshold, a positive scalar.
Logx	Logarithm base in the entropy formula, a positive scalar.

Outputs

XAp	Cross-approximate entropy estimates, a vector of length m+1. **The first value of **XAp** is the zeroth estimate, i.e. $-\Phi_1$, and the last value of **XAp** is the estimate for the specified **m**.
Phi	The number of matched state vectors for each embedding dimension from 0 to m+1.

References [1]

3.2.2 XSampEn: Cross-Sample Entropy

Syntax

```
[XSamp, A, B, Vcp_Ka_Kb] = XSampEn(Sig1, Sig2, 'm', 2, 'tau', 1, 'r', 0.2*std_pooled(Sig1,
Sig2), 'Logx', exp(1), 'Vcp', false)
XSamp, A, B, Vcp_Ka_Kb = XSampEn(Sig1, Sig2, m = 2, tau = 1, r = 0.2*std_pooled(Sig1,
Sig2), Logx = np.exp(1), Vcp = False)
XSamp, A, B, Vcp_Ka_Kb = XSampEn(Sig1, Sig2, m = 2, tau = 1, r = 0.2*std_pooled(Sig1,
Sig2), Logx = exp(1), Vcp = false)
```

Arguments

Sig1	First data sequence, a vector of > 10 elements.
Sig2	Second data sequence, a vector of > 10 elements.
m	Embedding dimension, a positive integer.
tau	Time delay, a positive integer.
r	Radius distance threshold, a positive scalar.
Logx	Logarithm base in the entropy formula, a positive scalar.
Vcp	Option to return the variance of the conditional probabilities and the number of overlapping matching vector pairs of lengths

Outputs

XSamp	Cross-sample entropy estimates, a vector of length m+1. **The first value of **XSamp** is the zeroth estimate, i.e. $Log(N1 * N2) - Log(A_1)$, and the last value of **XSamp** is the estimate for the specified **m**.
A	The number of matched state vectors for each embedding dimension from 0 to m.
B	The number of matched state vectors for each embedding dimension from 1 to m+1.
Vcp_Ka_Kb	A vector/tuple of three values corresponding to: **Vcp_Ka_Kb(1) = V_{cp}** Estimate of the variance of conditional probabilities used in the estimation of the sample entropy estimate. **Vcp_Ka_Kb(2) = K_a** The number of overlapping vector pairs of length m+1 **Vcp_Ka_Kb(3) = K_b** The number of overlapping vector pairs of length m

Note: Vcp is undefined for the zeroth embedding dimension (m = 0) and due to the computational demand, **will take substantially more time to return function outputs. See Appendix B in [3] for more info.

References [2] [3]

3.2.3 XFuzzEn: Cross-Fuzzy Entropy

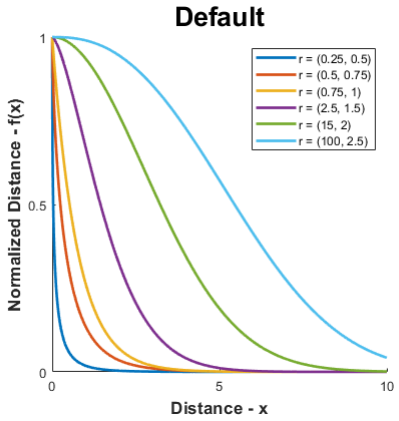
Syntax

```
[XFuzz, Ps1, Ps2] = XFuzzEn(Sig1, Sig2, 'm', 2, 'tau', 1, 'Fx', 'default, 'r', [0.2, 2], 'Logx', exp(1))
XFuzz, Ps1, Ps2 = XFuzzEn(Sig1, Sig2, m = 2, tau = 1, Fx = 'default', r = (0.2, 2), Logx = np.exp(1))
XFuzz, Ps1, Ps2 = XFuzzEn(Sig1, Sig2, m = 2, tau = 1, Fx = "default", r = (0.2, 2), Logx = exp(1))
```

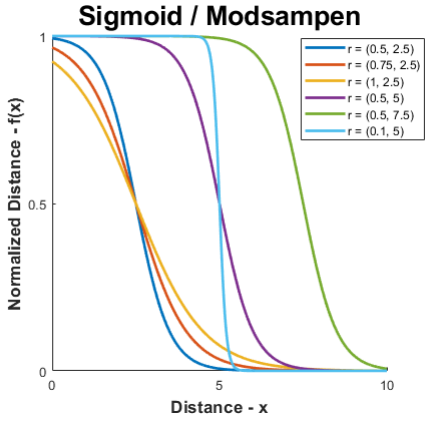
Arguments

Sig1 First data sequence, a vector of > 10 elements.
Sig2 Second data sequence, a vector of > 10 elements.
m Embedding dimension, a positive integer.
tau Time delay, a positive integer.
Fx Type of fuzzy function for distance transformation, one of the following strings:

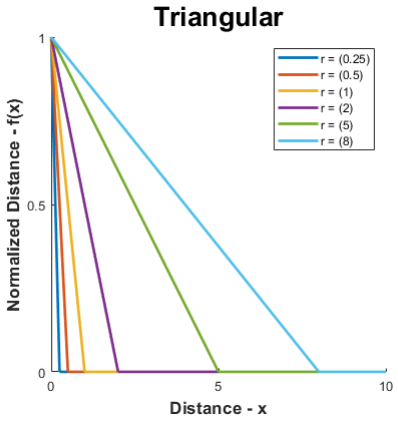
"default" $f(x) = exp(-\frac{x^{r_2}}{r_1})$



"sigmoid"/"modsampen" $f(x) = (1 + exp(\frac{x-r_2}{r_1}))^{-1}$

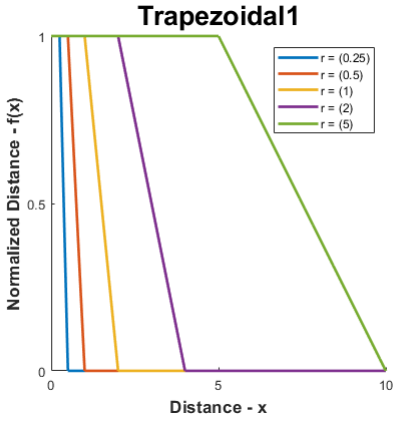


"triangular" $f(x) = \begin{cases} 1 - \frac{x}{r}, & x \leq r \\ 0, & x > r \end{cases}$



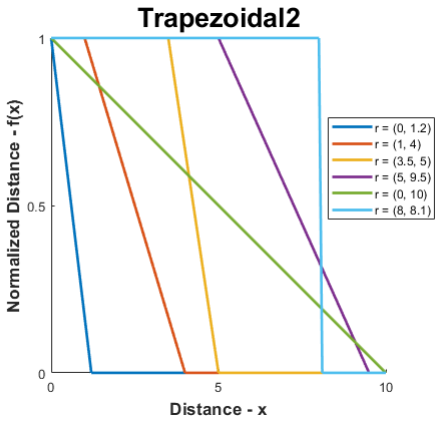
"trapezoidal1"

$$f(x) = \begin{cases} 1, & x < r \\ 2 - \frac{x}{r}, & r \leq x < 2r \\ 0, & x \geq 2r \end{cases}$$



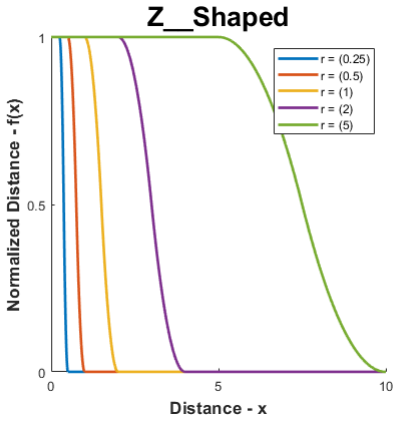
"trapezoidal2"

$$f(x) = \begin{cases} 1, & x < r \\ 1 - \frac{x - r_{min}}{r_{max} - r_{min}}, & r \leq x < r_{max} \\ 0, & x \geq r_{max} \end{cases}$$

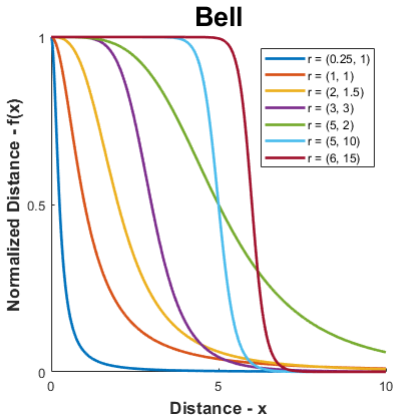


"z_shaped"

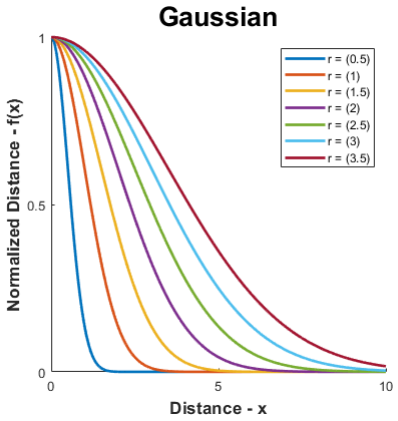
$$f(x) = \begin{cases} 1, & x < r \\ 1 - 2\left(\frac{x-r}{r}\right)^2, & r < x < \frac{3}{2}r \\ 2\left(\frac{x-r}{r}\right)^2, & \frac{3}{2}r < x < 2r \\ 0, & x > 2r \end{cases}$$



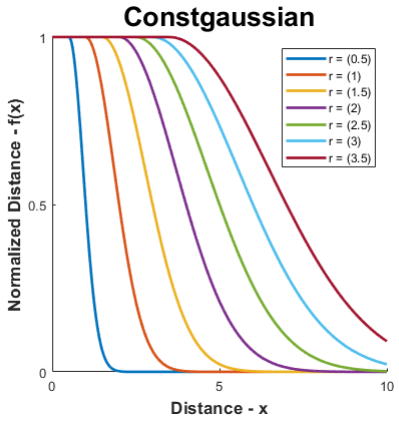
"bell" $f(x) = \frac{1}{1+\left|\frac{x}{r_1}\right|^{r_2}}$



"gaussian" $f(x) = exp(-\frac{x^2}{2r^2})$

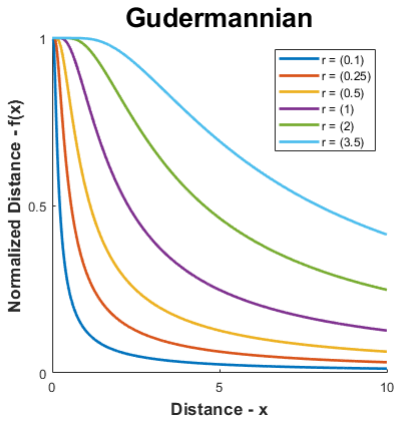


"constgaussian" $f(x) = \begin{cases} 1, & x < r_{min} \\ exp(-\log(2)\frac{(x-r)^2}{r}), & x \geq r_{max} \end{cases}$



"gudermannian"

$g(x) = atan(\frac{tanh(r_1)}{x})$
$f(x) = \frac{g(x)}{g(x_{max})}$
Note: Distances are normalized w.r.t. maximum distance relative to each state vector.



r Parameters of the fuzzy function specified by **Fx**, a 1 element scalar or a 2 element tuple of positive values depending on the fuzzy function as shown above.

Default	Two element tuple (or vector in MatLab)
Sigmoid/ModSampEn	Two element tuple (or vector in MatLab)
Trapezoidal2	Two element tuple (or vector in MatLab)
Bell	Two element tuple (or vector in MatLab)
Trapezoidal1	A scalar value
Triangular	A scalar value
Z_Shaped	A scalar value
Gaussian	A scalar value
ConstGaussian	A scalar value
Gudermannian	A scalar value

Logx Logarithm base in the entropy formula, a positive scalar.

Outputs

XFuzz Cross-fuzzy entropy estimates for each embedding dimension from 1 to m.
Ps1 The average fuzzy distances for embedding dimensions from 1 to m.
Ps2 The average fuzzy distances for embedding dimensions from 2 to m+1.

References

[46] [6]

3.2.4 XK2En: Cross-Kolmogorov Entropy

Syntax

```
[XK2, Ci] = XK2En(Sig1, Sig2, 'm', 2, 'tau', 1, 'r', 0.2*std_pooled(Sig1, Sig2), 'Logx', exp(1))
XK2, Ci = XK2En(Sig1, Sig2, m = 2, tau = 1, r = 0.2*std_pooled(Sig1, Sig2), Logx = np.exp(1))
XK2, Ci = XK2En(Sig1, Sig2, m = 2, tau = 1, r = 0.2*std_pooled(Sig1, Sig2), Logx = exp(1))
```

Arguments

Sig1	First data sequence, a vector of > 10 elements.
Sig2	Second data sequence, a vector of > 10 elements.
m	Embedding dimension, a positive integer.
tau	Time delay, a positive integer.
r	Distance threshold value, a positive scalar.
Logx	Logarithm base in the entropy formula, a positive scalar.

Outputs

XK2	Cross-Kolmogorov entropy estimates for each embedding dimension from 1 to m
Ci	The correlation sum for each embedding dimension from 1 to m.

References [83]

3.2.5 XPermEn: Cross-Permutation Entropy

Syntax

```
[XPerm] = XPermEn(Sig1, Sig2, 'm', 3, 'tau', 1, 'Logx', 2)
XPerm = XPermEn(Sig1, Sig2, m = 3, tau = 1, Logx = 2)
XPerm = XPermEn(Sig1, Sig2, m = 3, tau = 1, Logx = 2)
```

Arguments

Sig1	First data sequence, a vector of > 10 elements.
Sig2	Second data sequence, a vector of > 10 elements. **NOTE: Sig1** and **Sig2** must have the same number of elements.
m	Embedding dimension, an integer > 2. **NOTE: XPerm** is undefined for **m** < 3.
tau	Time delay, a positive integer.
Logx	Logarithm base in the entropy formula, a positive scalar. (Enter 0 for natural logarithm)

Outputs

XPerm	Cross-permutation entropy estimate.

References [47]

3.2.6 XCondEn: Cross-Conditional Entropy

Syntax

```
[XCond, SEw, SEz] = XCondEn(Sig1, Sig2, 'm', 2, 'tau', 1, 'c', 6, 'Logx', exp(1), 'Norm', false)
XCond, SEw, SEz = XCondEn(Sig1, Sig2, m = 2, tau = 1, c = 6, Logx = np.exp(1), Norm = False)
XCond, SEw, SEz = XCondEn(Sig1, Sig2, m = 2, tau = 1, c = 6, Logx = exp(1), Norm = false)
```

Arguments

Sig1	First data sequence, a vector of > 10 elements.
Sig2	Second data sequence, a vector of > 10 elements. **NOTE:** **Sig1** and **Sig2** must have the same number of elements. **NOTE:** **XCondEn** is direction-dependent. Therefore, the order of the data sequences in **Sig** matters. If the first column of **Sig** is the sequence 'y', and the second column is the sequence 'u', **XCond** is the amount of information carried by y(i) when the pattern u(i) is found.
m	Embedding dimension, an integer > 1.
tau	Time delay, a positive integer.
c	Number of symbols in symbolic transformation, in integer > 1
Logx	Logarithm base in the entropy formula, a positive scalar.
Norm	Normalization of **XCond** value: `true` no normalisation (default) `false` normalises w.r.t cross-Shannon entropy.

Outputs

XCond	Corrected Cross-Conditional entropy estimate.
SEw	Cross-Shannon entropy estimate for **m**.
SEz	Cross-Shannon entropy estimate for **m+1**.

References [18]

3.2.7 XDistEn: Cross-Distribution Entropy

Syntax

```
[XDist, Ppi] = XDistEn(Sig1, Sig2, 'm', 2, 'tau', 1, 'Bins', 'sturges', 'Logx', 2, 'Norm', true)
XDist, Ppi = XDistEn(Sig1, Sig2, m = 2, tau = 1, Bins = 'sturges', Logx = 2, Norm = True)
XDist, Ppi = XDistEn(Sig1, Sig2, m = 2, tau = 1, Bins = "sturges", Logx = 2, Norm = true)
```

Arguments

Sig1	First data sequence, a vector of > 10 elements.
Sig2	Second data sequence, a vector of > 10 elements.
m	Embedding dimension, an integer > 1.
tau	Time delay, a positive integer.
Bins	Histogram bin selection method, in integer > 1 indicating the number of bins, or one of the following strings: "sturges", "sqrt", "rice", "doanes" [default: "sturges"] >>> More info on binning methods.
Logx	Logarithm base in the entropy formula, a positive scalar. (Enter 0 for natural logarithm)
Norm	Normalization of **XDist** value: `false` no normalisation `true` normalises w.r.t number of histogram bins (default)

Outputs

XDist	Cross-Distribution entropy estimate.
Ppi	Probability of each histogram bin.

References

[19]

3.2.8 XSpecEn: Cross-Spectral Entropy

Syntax

```
[XSpec, BandEn] = XSpecEn(Sig1, Sig2, 'N', 2*max(length(Sig1), length(Sig2))+1,
'Freqs', [0,1], 'Logx', exp(1), 'Norm', true)
XSpec, BandEn = XSpecEn(Sig1, Sig2, N = 2*np.maximum(len(Sig1),len(Sig2)) + 1,
Freqs = (0,1), Logx = np.exp(1), Norm = True)
XSpec, BandEn = XSpecEn(Sig, N = 2*max(length(Sig1),length(Sig2)) + 1, Freqs
= (0,1), Logx = exp(1), Norm = true)
```

Arguments

Sig1	First data sequence, a vector of > 10 elements.
Sig2	Second data sequence, a vector of > 10 elements.
N	Resolution of the N-point fft, an integer > 1.
Freqs	Normalised band edge-frequencies for calculating the band entropy (BandEn), a 2 element tuple with values in range [0, 1] where 1 is the Nyquist frequency. * When no edge frequencies are provided, BandEn==XSpecEn
Logx	Logarithm base in the entropy formula, a positive scalar.
Norm	Normalization of **XSpec** value: `false` no normalisation `true` normalises **XSpec** w.r.t number of Nyquist frequency values, and **BandEn** w.r.t. range of frequencies in the band given by **Freqs**. (default)

Outputs

XSpec	Cross-Spectral entropy estimate.
BandEn	Cross-Spectral band entropy estimate.

References [83]

Note: In contrast to other *Cross*-entropies, cross-spectral entropy (XSpecEn) is not derived from information theory or dynamical systems theory, and instead is an estimate of the frequency cross-spectrum curve estimated using the discrete time Fourier transform.

3.3 Multivariate Entropy Functions

Multivariate Data

Unlike *Base* and *Cross-* entropy functions, ***Multivariate*** entropy functions operate on a matrix of univariate sequences. This, in the following functions, the first argument **`Data`** is an N x M matrix representing M (> 1) univariate sequences of N (> 10) samples.

ATTENTION

By default, the **`MvSampEn`** and **`MvFuzzEn`** multivariate entropy algorithms estimate entropy values using the "full" method by comparing delay vectors across all possible m+1 expansions of the embedding space as applied in [79] [80] [81]. These methods are **not lower-bounded to 0**, like most entropy algorithms, so **`MvMSEn`** may return negative entropy values if the base multivariate entropy function is **`MvSampEn`** and **`MvFuzzEn`**, even for stochastic processes...

3.3.1 MvSampEn: Multivariate Sample Entropy

Syntax

```
[MSamp, B0, Bt, B1] = MvSampEn(Data, 'm', 2*ones(1,size(Data,2)),
   'tau', ones(1,size(Data,2)), 'r', 0.2, 'Norm', false, 'Logx', exp(1))
MSamp, B0, Bt, B1 = MvSampEn(Data, m = 2*np.ones(Data.shape[1]),
   tau = np.ones(Data.shape[1]), r = 0.2, Norm = False, Logx = np.exp(1))
MSamp, B0, Bt, B1 = MvSampEn(Data, m = 2*ones(size(Data,2)),
   tau = ones(size(Data,2)), r = 0.2, Norm = false Logx = exp(1))
```

NOTE

To maximize the number of points in the embedding process, this algorithm uses **`N-max(m*tau)`** delay vectors and **not `N-max(m)*max(tau)`** as employed in [80],[79].

Arguments

`Data`	Multivariate dataset, NxM matrix of N (> 10) observations (rows) and M (> 1) univariate data sequences (cols).
`m`	Embedding Dimension, a vector of M positive integers.
`tau`	Time Delay, a vector of M positive integers.
`r`	Distance threshold, a positive scalar.
`Norm`	Normalisation of all M sequences to unit variance, a boolean
`Logx`	Logarithm base in the entropy formula, a positive scalar.

Outputs

`MSamp`	Multivariate sample entropy estimates.
`B0`	The number of matched state vectors for given m embedding dimensions.
`Bt`	The number of matched state vectors for the joint total m+1 embedded subspace.
`B1`	The number of matched state vectors for each possible m+1 embedded subspace.

References [79] [80]

3.3.2 MvFuzzEn: Multivariate Fuzzy Entropy

Syntax

```
[MFuzz, B0, Bt, B1] = MvFuzzEn(Data, 'm', 2*ones(1,size(Data,2)),
   'tau', ones(1,size(Data,2)), 'Fx', 'default', 'r', [0.2,2],
   'Norm', false, 'Logx', exp(1))
MFuzz, B0, Bt, B1 = MvFuzzEn(Data, m = 2*np.ones(Data.shape[1]),
   tau = np.ones(Data.shape[1]), Fx = 'default', r = (0.2,2),
   Norm = False, Logx = np.exp(1))
MFuzz, B0, Bt, B1 = MvFuzzEn(Data, m = 2*ones(size(Data,2)),
   tau = ones(size(Data,2)), Fx = "default", r = (0.2,2),
   Norm = false, Logx = exp(1))
```

IMPORTANT

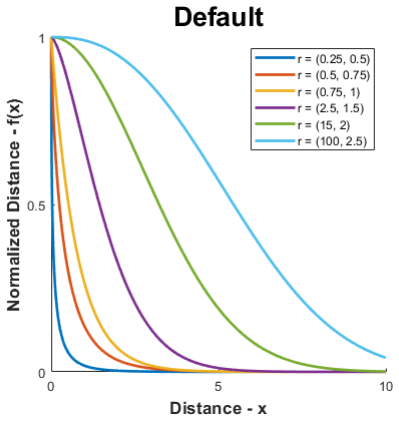
The entropy value returned as **MFuzz** is estimated using the "full" method [i.e. `-log(Bt/B0)`] which compares delay vectors across all possible m+1 expansions of the embedding space as applied in [79]. Contrary to conventional definitions of fuzzy entropy, this method does not provide a lower bound of 0!! Thus, it is possible to obtain negative entropy values for multivariate fuzzy entropy, even for stochastic processes...
Alternatively, one can calculate **MFuzz** via the "naive" method, which ensures a lower bound of 0, by using the average vector distances for an individual m+1 subspace (B1) [e.g. `-log(B1(1)/B0)`], or the average for all m+1 subspaces [i.e. `-log(mean(B1)/B0)`].

NOTE

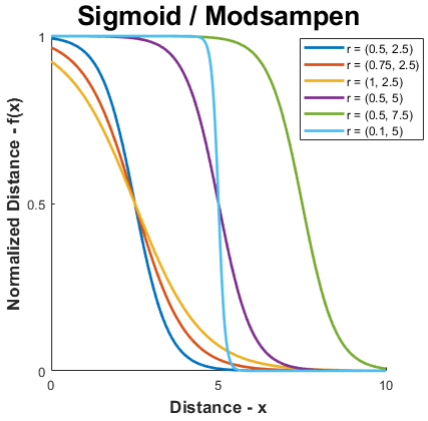
To maximize the number of points in the embedding process, this algorithm uses **N-max(m*tau)** delay vectors and **not N-max(m)*max(tau)** as employed in [80],[79].

Arguments

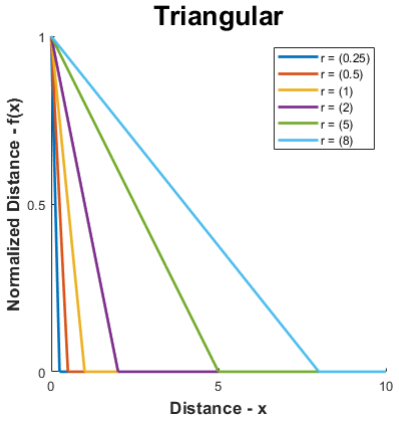
Data	Multivariate dataset, NxM matrix of N (> 10) observations (rows) and M (> 1) univariate data sequences (cols).	
m	Embedding Dimension, a vector of M positive integers.	
tau	Time Delay, a vector of M positive integers.	
Fx	Type of fuzzy function for distance transformation, one of the following strings:	
	"default"	$f(x) = exp(-\frac{x^{r_2}}{r_1})$



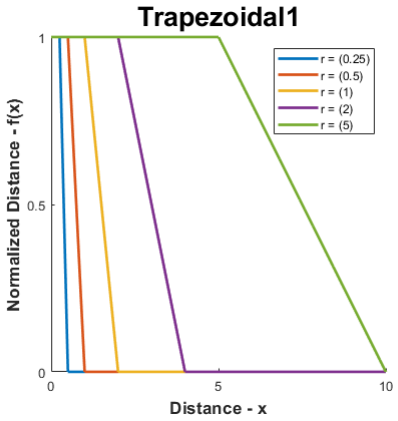
"sigmoid"/"modsampen" $f(x) = (1 + exp(\frac{x-r_2}{r_1}))^{-1}$



"triangular" $f(x) = \begin{cases} 1 - \frac{x}{r}, & x \leq r \\ 0, & x > r \end{cases}$

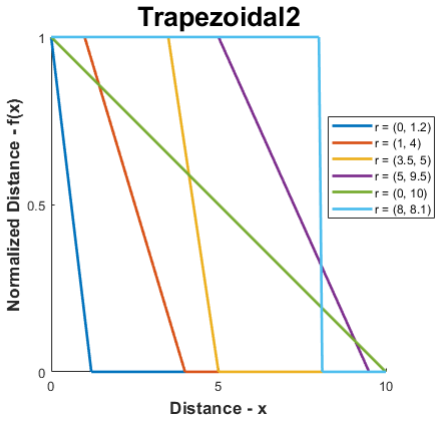


"trapezoidal1" $f(x) = \begin{cases} 1, & x < r \\ 2 - \frac{x}{r}, & r \leq x < 2r \\ 0, & x \geq 2r \end{cases}$



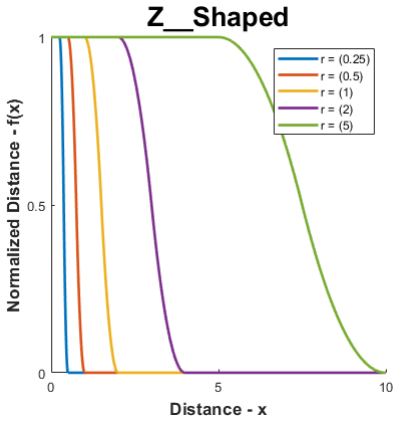
"trapezoidal2"

$$f(x) = \begin{cases} 1, & x < r \\ 1 - \frac{x - r_{min}}{r_{max} - r_{min}}, & r \leq x < r_{max} \\ 0, & x \geq r_{max} \end{cases}$$



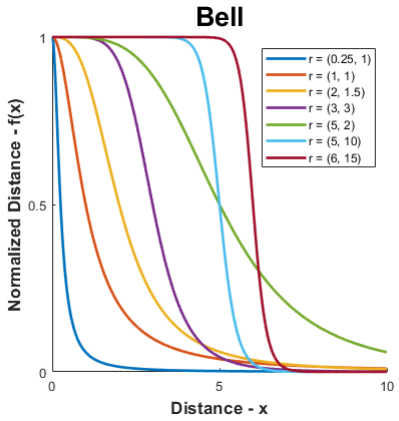
"z_shaped"

$$f(x) = \begin{cases} 1, & x < r \\ 1 - 2\left(\frac{x-r}{r}\right)^2, & r < x < \frac{3}{2}r \\ 2\left(\frac{x-r}{r}\right)^2, & \frac{3}{2}r < x < 2r \\ 0, & x > 2r \end{cases}$$



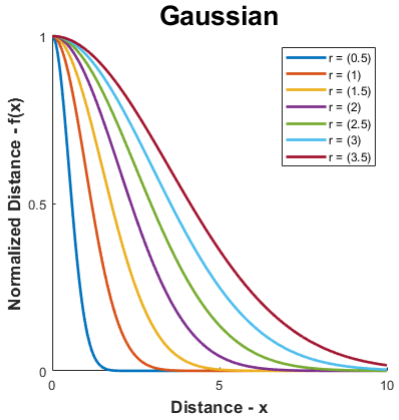
"bell"

$$f\left(x\right) = \frac{1}{1 + \left|\frac{x}{r_1}\right|^{r_2}}$$



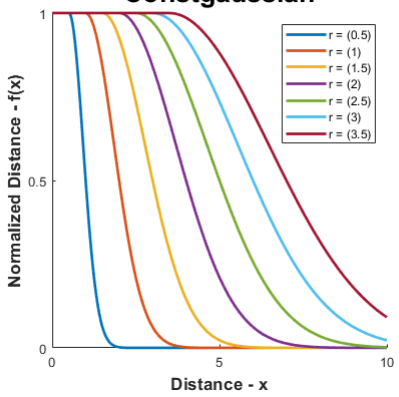
"gaussian"

$f(x) = exp(-\frac{x^2}{2r^2})$



"constgaussian"

$$f(x) = \begin{cases} 1, & x < r_{min} \\ exp(-\log(2)\frac{(x-r)^2}{r}), & x \geq r_{max} \end{cases}$$

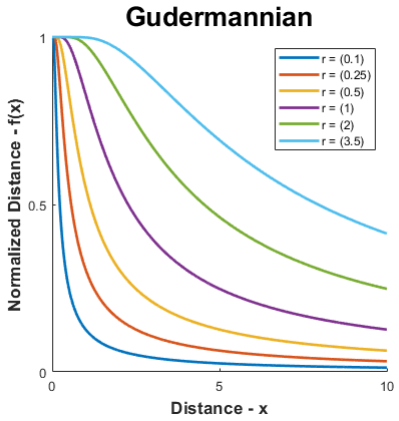


"gudermannian"

$g(x) = atan(\frac{tanh(r_1)}{x})$

$f(x) = \frac{g(x)}{g(x_{max})}$

Note: Distances are normalized w.r.t. maximum distance relative to each state vector.



r	Parameters of the fuzzy function specified by **Fx**, a 1 element scalar or a 2 element tuple of positive values depending on the fuzzy function as shown above.	
	Default	Two element tuple (or vector in MatLab)
	Sigmoid/ModSampEn	Two element tuple (or vector in MatLab)
	Trapezoidal2	Two element tuple (or vector in MatLab)
	Bell	Two element tuple (or vector in MatLab)
	Trapezoidal1	A scalar value
	Triangular	A scalar value
	Z_Shaped	A scalar value
	Gaussian	A scalar value
	ConstGaussian	A scalar value
	Gudermannian	A scalar value
Norm	Normalisation of all M sequences to unit variance, a boolean	
Logx	Logarithm base in the entropy formula, a positive scalar.	

Outputs

MFuzz	Multivariate fuzzy entropy estimates.
B0	Average delay vector distance for given m embedding dimensions.
Bt	Average delay vector distance for the joint total m+1 embedded subspace.
B1	Average delay vector distance for each possible m+1 embedded subspace.

References

[81]

3.3.3 MvDispEn: Multivariate Dispersion Entropy

Syntax

```
[MDisp, RDE] = MvDispEn(Data, 'm', 2*ones(1,size(Data,2)),
   'tau', ones(1,size(Data,2)), 'c', 3, 'Typex', 'NCDF', 'Methodx', 'v1',
   'Norm', false, 'Logx', exp(1))
MDisp, RDE = MvDispEn(Data, m = 2*np.ones(Data.shape[1]),
   tau = np.ones(Data.shape[1]), c = 3, Typex = 'NCDF', Methodx = 'v1',
   Norm = False, Logx = np.exp(1))
MDisp, RDE = MvDispEn(Data, m = 2*ones(size(Data,2)),
   tau = ones(size(Data,2)), c = 3, Typex = "NCDF", Methodx = "v1",
   Norm = false, Logx = exp(1))
```

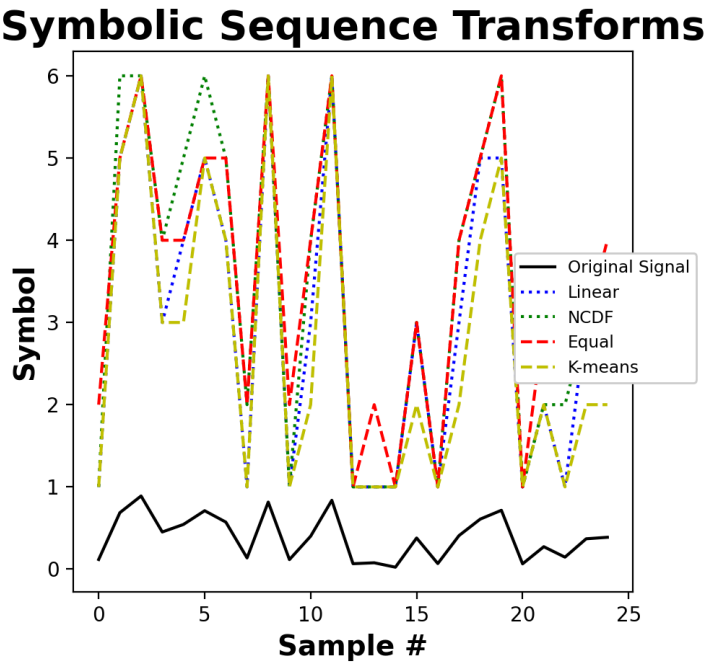
IMPORTANT

By default, `MvDispEn` uses the method termed ***mvDEii*** in [82], which follows the original multivariate embedding approach of Ahmed & Mandic [79], [80]. The ***v1*** method therefore returns a singular entropy estimate.

If the ***v2*** method is selected (`Methodx=='v2'`), the main method outlined in [82] termed ***mvDE*** is applied. In this case, entropy is estimated using each combination of multivariate delay vectors with lengths 1:max(m), with each entropy value returned accordingly. See [82] for more info.

Arguments

`Data`	Multivariate dataset, NxM matrix of N (> 10) observations (rows) and M (> 1) univariate data sequences (cols).	
`m`	Embedding Dimension, a vector of M positive integers.	
`tau`	Time Delay, a vector of M positive integers.	
`c`	Number of symbols in transform, an integer > 1.	
`Typex`	Type of symbolic sequence transform, one of the following strings:	
	`"ncdf"`	Normalised cumulative distribution function [22]
	`"kmeans"`	K-means clustering algorithm. **Note: The *"kmeans"* algorithm uses random initialization conditions. This causes results to vary slightly each time it is called.
	`"linear"`	Linear segmentation of signal range
	`"equal"`	Approx. equal number of symbols.



Methodx The method of multivariate dispersion entropy estimation as outlined in [?], either:

- **"v1"** employs the method consistent with the original multivariate embedding approach of Ahmed & Mandic [?], termed **mvDEii** in [?]. (default)
- **"v2"** employs the main method derived in [?], termed **mvDE**.

Norm Normalisation of MDisp value w.r.t. the number of unique dispersion patterns, a boolean

Logx Logarithm base in the entropy formula, a positive scalar.

Outputs

MDisp Multivariate dispersion entropy estimates.

RDE Multivariate reverse dispersion entropy estimate.

References

[82]

3.3.4 MvPermEn: Multivariate Permutation Entropy

Syntax

```
[MPerm, MPnorm] = MvPermEn(Data, 'm', 2*ones(1,size(Data,2)),
   'tau', ones(1,size(Data,2)), 'Typex', [], 'tpx', -1,
   'Norm', false, 'Logx', 2)
MPerm, MPnorm = MvPermEn(Data, m = 2*np.ones(Data.shape[1]),
   tau = np.ones(Data.shape[1]), Typex = None, tpx = -1,
   Norm = False, Logx = 2)
MPerm, MPnorm = MvPermEn(Data, m = 2*ones(size(Data,2)),
   tau = ones(size(Data,2)), Typex = nothing, tpx = -1,
   Norm = false, Logx = 2)
```

IMPORTANT

The multivariate permutation entropy algorithm implemented here uses multivariate embedding based on Takens' embedding theorem, and follows the methods for multivariate entropy estimation through shared spatial reconstruction as originally presented by Ahmed & Mandic [79] [80].
This function does **NOT** use the multivariate permutation entropy algorithm of Morabito et al. (Entropy, 2012) where the entropy values of individual univariate sequences are averaged because such methods do not follow the definition of multivariate embedding and therefore do not consider cross-channel statistical complexity.

NOTE

To maximize the number of points in the embedding process, this algorithm uses **`N-max(m*tau)`** delay vectors and **not `N-max(m)*max(tau)`** as employed in [80],[79].

Arguments

`Data`	Multivariate dataset, NxM matrix of N ($>$ 10) observations (rows) and M ($>$ 1) univariate data sequences (cols).	
`m`	Embedding Dimension, a vector of M positive integers.	
`tau`	Time Delay, a vector of M positive integers.	
`Typex`	Variant of permutation entropy, one of the following strings:	
	`"modified"`	Modified permutation entropy [11]
	`"weighted"`	Weighted permutation entropy [12]
	`"ampaware"`	Amplitude-aware permutation entropy [13]
	`"edge"`	Edge permutation entropy [14]
	`"phase"`	Phase permutation entropy [17]
`tpx`	Tuning parameter for the permutation entropy specified by the **`Typex`** argument.	
	`ampaware`	**`tpx`** is the A parameter, a value in range [0 1] (default: 0.5)
	`edge`	**`tpx`** is the r sensitivity parameter, a scalar $>$ 0 (default: 1)
	`phase`	**`tpx`** is the option to unwrap the phase shift of the Hilbert-transformed data sequence, either [ ] or 1 (default: [ ]).
`Norm`	Normalisation of **`MPnorm`** value, a boolean operator:	
	`false`	normalises w.r.t *Log*(# of permutation symbols [`sum(m)`]) - default
	`true`	normalises w.r.t *Log*(# of all possible permutations [`sum(m)!`])

Logx	Logarithm base in the entropy formula, a positive scalar.

Outputs

MPerm	Multivariate permutation entropy estimate
MPnorm	Normalized multivariate permutation entropy estimates.

References

[83] [79]

3.3.5 MvCoSiEn: Multivariate Cosine Similarity Entropy

Syntax

```
[MCoSi, Bm] = MvCoSiEn(Data, 'm', 2*ones(1,size(Data,2)),
   'tau', ones(1,size(Data,2)), 'r', 0.1, 'Logx', 2, 'Norm', 0)
MCoSi, Bm = MvCoSiEn(Data, m = 2*np.ones(Data.shape[1]),
   tau = np.ones(Data.shape[1]), r = 0.1, Logx = 2, Norm = 0)
MCoSi, Bm = MvCoSiEn(Data, m = 2*ones(size(Data,2)),
   tau = ones(size(Data,2)), r = 0.1, Logx = 2, Norm = 0)
```

NOTE

To maximize the number of points in the embedding process, this algorithm uses **`N-max(m*tau)`** delay vectors and **not `N-max(m)*max(tau)`** as employed in [78],[79].

Arguments

`Data`	Multivariate dataset, NxM matrix of N (> 10) observations (rows) and M (> 1) univariate data sequences (cols).
`m`	Embedding Dimension, a vector of M positive integers.
`tau`	Time Delay, a vector of M positive integers.
`r`	Angular threshold, a value in range [0 < **`r`** < 1]
`Logx`	Logarithm base in the entropy formula, a positive scalar.
`Norm`	Normalisation of each sequence in **`Data`**, an integer in range [0 4]: 0 - no normalisation (default) 1 - median removed 2 - mean removed 3 - normalised by unit variance 4 - normalised to range [-1 1]

Outputs

`MCoSi`	Multivariate cosine similarity entropy estimate.
`Bm`	Sum of global probabilities.

References [32] [78]

3.4 Multiscale Entropy Functions

A key advantage of the EntropyHub toolkit is that so many variants of multiscale entropy can be easily calculated using any of the **`Base`** entropy functions. This is achieved using a multiscale entropy object (**`Mobj`**), returned by **`MSobject()`**, to specify the type of entropy and its parameters.

> Multiscale entropy functions have two positional arguments:
> the time series signal **`Sig`**, and
> the multiscale entropy object, **`Mobj`**.

Examples (shown in Julia syntax):

Original multiscale entropy [48]

```
Mobj = MSobject(SampEn)
mse = MSEn(Sig, Mobj)
```

Time-shifted multiscale approximate entropy with varying tolerance across scales [49]

```
Mobj = MSobject(ApEn, m = 5, r = 0.25)
mse = MSEn(Sig, Mobj, Methodx = "timeshift", RadNew = 1)
```

Composite multiscale conditional entropy with a 10-symbol data sequence, calculated up to 5 temporal scales [60]

```
Mobj = MSobject(CondEn, m = 5, c = 10)
cmse = cMSEn(Sig, Mobj, scales = 5)
```

Refined-Composite multiscale entropy calculated in bits [61]

```
Mobj = MSobject(SampEn, Logx = 2)
rcmse = cMSEn(Sig, Mobj, Refined = true)
```

Refined multiscale fuzzy entropy calulated using a sigmoidal fuzzy function and a time delay of 4

```
Mobj = MSobject(FuzzEn, tau = 4, Fx = "sigmoid")
rmse = rMSEn(Sig, Mobj)
```

Hierarchical multiscale edge permutation entropy with an 'r' sensitivity parameter = 2.66 normalized w.r.t. the number of symbols (4), and calculated up to 5 hierarchical scales

```
Mobj = MSobject(PermEn, m = 4, Typex = "edge", tpx = 2.66, Norm = true)
hmse = hMSEn(Sig, Mobj, scale = 5)
```

3.4.1 MSobject: Multiscale Entropy Object

Syntax

```
Mobj = MSobject(EnType, varargin)
Mobj = MSobject(EnType, **kwargs)
Mobj = MSobject(EnType::Function, kwargs...)
```

Arguments

EnType	In **MatLab** and **Python**, **EnType** is a case-sensitive string corresponding to a valid **Base** or **Cross-** entropy function, e.g.'SyDyEn' or 'XDistEn', etc. In **Julia**, **EnType** is a **Base** or **Cross-** entropy Function object, e.g. EntropyHub.ApEn (or ApEn if imported independently), or XSpecEn, etc.
varargin ****kwargs** **kwargs...**	Any valid keyword arguments (Name/Value pairs) for the entropy function specified by **EnType**

Outputs

Mobj	Multiscale Entropy object.

3.4.2 MSEn: Multiscale Entropy

Syntax

```
[MSx, Ci] = MSEn(Sig, Mobj, 'Scales', 3, 'Methodx', 'coarse', 'RadNew', 0, 'Plotx', false)
MSx, Ci = MSEn(Sig, Mobj, Scales = 3, Methodx = 'coarse', RadNew = 0, Plotx = False)
MSx, Ci = MSEn(Sig, Mobj, Scales = 3, Methodx = "coarse", RadNew = 0, Plotx = false)
```

Arguments

Sig Time series signals, a vector of length > 10.

Mobj Multiscale Entropy object created with **MSobject()** - 3.4.1

Scales Number of grained time scales, a positive integer.

Methodx Type of graining method, one of the following strings:

"coarse" [48]

$$y_j^{(\tau)} = \frac{1}{\tau} \sum_{i=(j-1)\tau+1}^{j\tau} x_i, \qquad 1 \le j \le \frac{N}{\tau}$$

"generalized" [59]

$$y_j^{(\tau)} = \frac{1}{\tau} \sum_{k=0}^{\tau-1} (x_{j-k} - \bar{x})^2, \qquad 1 \le j \le N - \tau + 1$$

"modified"

$$y_j^{(\tau)} = \frac{1}{\tau} \sum_{k=0}^{\tau-1} x_{j-k}, \qquad 1 \le j \le N - \tau + 1$$

"timeshift" [57]

$$y_\beta^\tau = (x_\beta, x_{\beta+\tau}, x_{\beta+2\tau}, ..., x_{\beta+\lfloor \frac{N-\beta}{\tau} \rfloor \tau}) \qquad for \quad \beta = 1, 2, ..., \tau$$

$$TSME_\tau = \frac{1}{\tau} \sum_{\beta=1}^{\tau} F_{EnType}(y_\beta^\tau)$$

"imf" Grained time series at scale τ is the cumulative sum of intrinsic mode functions (IMF^1 to IMF^τ), where IMF^1 is the first sifting. [53]
***Note:** The empirical mode decomposition method use to derive the IMFs differs slightly between **MatLab, Python** and **Julia**, so **MSx** values will be inconsistent between the programming environments.
****Note:** **Julia's** empirical mode decomposition method is unstable and may not fully decompose highly stochastic or aperiodic signals.

RadNew Radius rescaling method, an integer in the range [0 4].
When the **Base** entropy method specified by **Mobj** is **SampEn** or **ApEn**, **RadNew** allows the radius threshold to be updated based on the grained signal at each time scale (X_τ). If a radius threshold value **(r)** is specified in **Mobj**, this becomes the rescaling coefficient, otherwise it is set to 0.2 (default). The value of **RadNew** specifies one of the following methods:

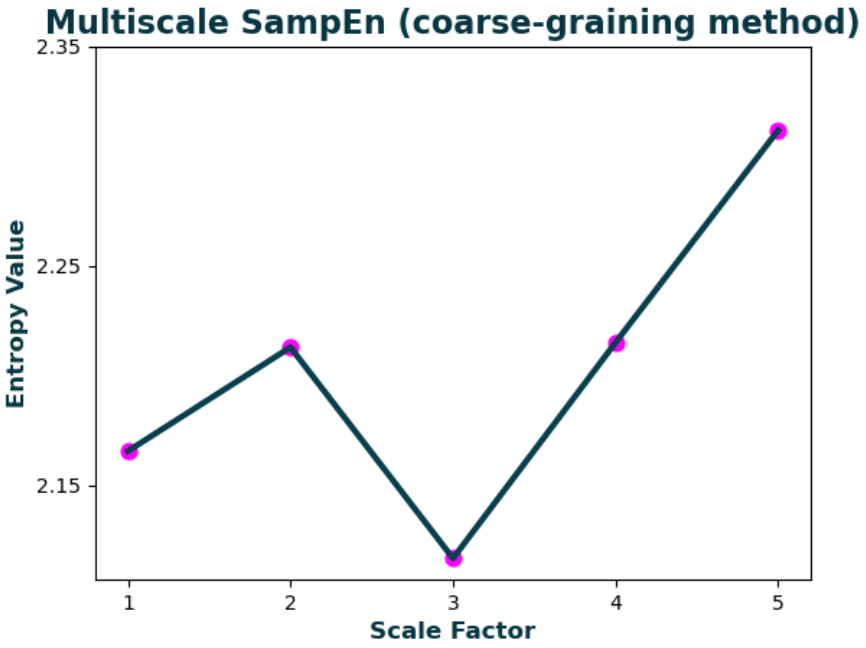
0 no rescaling

1	Standard Deviation - $r\sigma_{X_\tau}$
2	Variance - $r\sigma^2_{X_\tau}$
3	Mean Absolute Deviation - $r(\frac{1}{N}\sum\|X_\tau - \bar{X}_\tau\|)$
4	Median Absolute Deviation - $r(median(\|X_\tau - median(X_\tau)\|))$

`Plotx` A plot of the multiscale entropy curve

`true`	Plots time scale vs entropy value.
`false`	No plot.

An example multiscale entropy curve of a normally distributed random number sequence using sample entropy over 5 coarse-grained time scales.



Outputs

`MSx` Multiscale entropy estimate at each time scale (τ), a vector of length **`Scales`**.

`Ci` Complexity index (area under the multiscale entropy curve), a scalar.

References

[48] [49] [50] [51] [52] [53] [54] [55] [56] [57] [58] [59]

3.4.3 cMSEn: Composite & Refined-Composite Multiscale Entropy

Syntax

```
[MSx, Ci] = cMSEn(Sig, Mobj, 'Scales', 3, 'RadNew', 0, 'Refined', false, 'Plotx', false)
MSx, Ci = cMSEn(Sig, Mobj, Scales = 3, RadNew = 0, Refined = False, Plotx = False)
MSx, Ci = cMSEn(Sig, Mobj, Scales = 3, RadNew = 0, Refined = false, Plotx = false)
```

Arguments

Sig Time series signals, a vector of length > 10.

Mobj Multiscale Entropy object created with **MSobject()** - 3.4.1

Scales Number of time scales, a positive integer.

RadNew Radius rescaling method, an integer in the range [0 4].
When the **Base** entropy method specified by **Mobj** is **SampEn** or **ApEn**, **RadNew** allows the radius threshold to be updated based on the grained signal at each time scale (X_τ). If a radius threshold value **(r)** is specified in **Mobj**, this becomes the rescaling coefficient, otherwise it is set to 0.2 (default). The value of **RadNew** specifies one of the following methods:

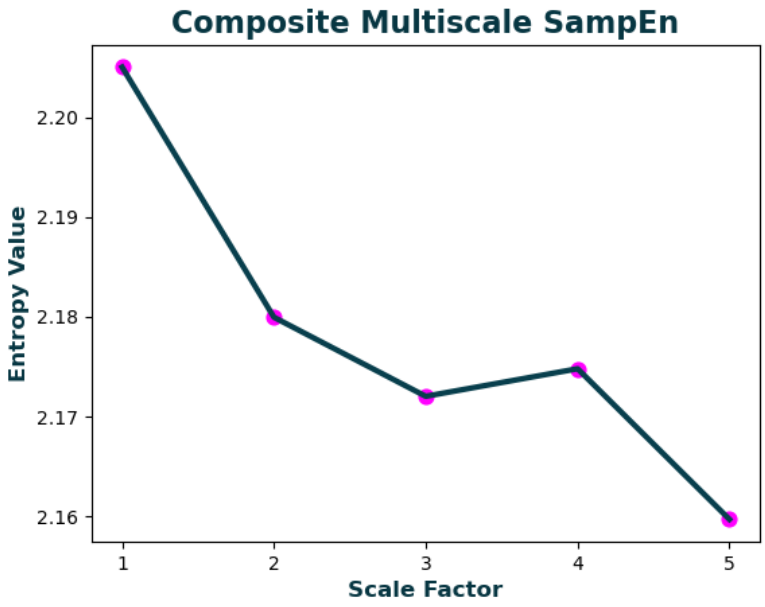
0	no rescaling
1	Standard Deviation - $r\sigma_{X_\tau}$
2	Variance - $r\sigma^2_{X_\tau}$
3	Mean Absolute Deviation - $r(\frac{1}{N}\sum\|X_\tau - \bar{X}_\tau\|)$
4	Median Absolute Deviation - $r(median(\|X_\tau - median(X_\tau)\|))$

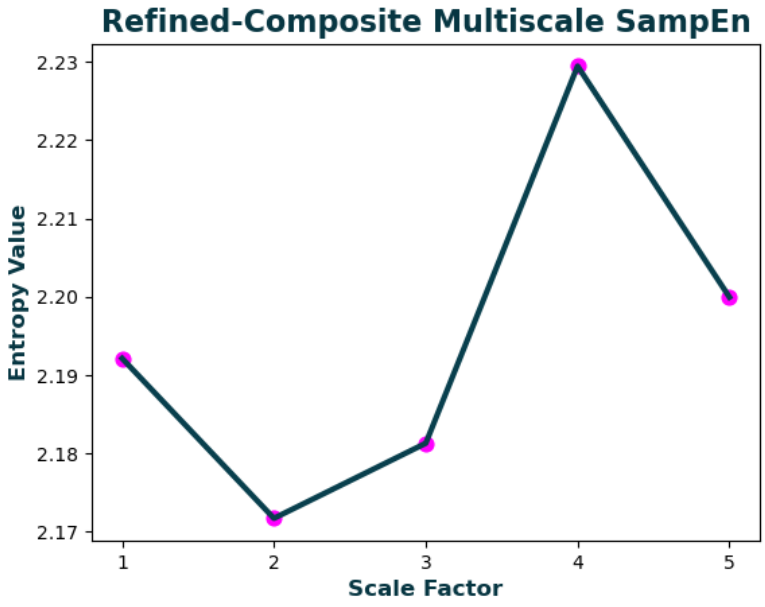
Refined When **Refined == true** and the entropy function (**EnType**) contained in **Mobj** is either **SampEn**, or **FuzzEn**, **cMSEn** returns the refined-composite multiscale entropy (**rcMSEn**). [61] [62]

Plotx A plot of the multiscale entropy curve

`true`	Plots time scale vs entropy value.
`false`	No plot.

Example of composite multiscale entropy and refined-composite multiscale entropy curves for normally distributed random number sequences using sample entropy over 5 time scales.





Note: refined-composite multiscale sample entropy uses moving averaging to perform graining (analogous to the `modified` graining method in **`MSEn`**), but refined-composite multiscale fuzzy entropy uses the moving variance (analogous to the `generalized` graining method in **`MSEn`**).

Outputs

`MSx`	Composite multiscale entropy estimate at each time scale (τ), a vector of length **`Scales`**.
`Ci`	Complexity index (area under the multiscale entropy curve), a scalar.

References [48] [49] [50] [60] [61]

3.4.4 rMSEn: Refined Multiscale Entropy

Syntax

```
[MSx, Ci] = rMSEn(Sig, Mobj, 'Scales', 3, 'F_Order', 6, 'F_Num', 0.5, 'RadNew', 0, 'Plotx', false)
MSx, Ci = rMSEn(Sig, Mobj, Scales = 3, F_Order = 6, F_Num = 0.5, RadNew = 0, Plotx = False)
MSx, Ci = rMSEn(Sig, Mobj, Scales = 3, F_Order = 6, F_Num = 0.5, RadNew = 0, Plotx = false)
```

Arguments

Sig Time series signals, a vector of length > 10.

Mobj Multiscale Entropy object created with **MSobject()** - 3.4.1

Scales Number of time scales, an integer > 0.

F_Order Butterworth low-pass filter order, a positive integer > 1, (default: 6)

F_Num Numerator of Butterworth low-pass filter cutoff frequency, where $[0 < F_{Num} < 1]$. The cutoff frequency at each scale (τ) becomes: $F_c = \frac{F_{Num}}{\tau}$ (default: 0.5)

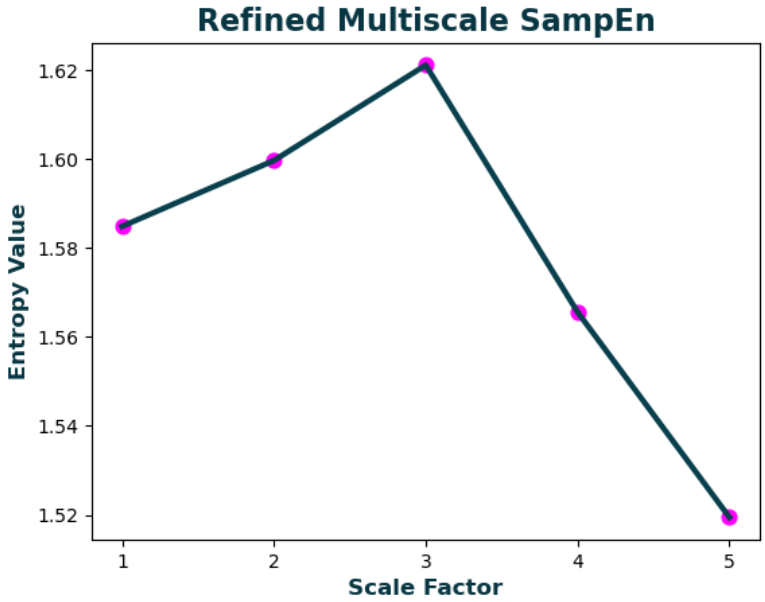
RadNew Radius rescaling method, an integer in the range [0 4].
When the **Base** entropy method specified by **Mobj** is **SampEn** or **ApEn**, **RadNew** allows the radius threshold to be updated based on the grained signal at each time scale (X_τ). If a radius threshold value **(r)** is specified in **Mobj**, this becomes the rescaling coefficient, otherwise it is set to 0.2 (default). The value of **RadNew** specifies one of the following methods:

0	no rescaling
1	Standard Deviation - $r\sigma_{X_\tau}$
2	Variance - $r\sigma^2_{X_\tau}$
3	Mean Absolute Deviation - $r(\frac{1}{N}\sum\|X_\tau - \bar{X_\tau}\|)$
4	Median Absolute Deviation - $r(median(\|X_\tau - median(X_\tau)\|))$

Plotx A plot of the multiscale entropy curve

`true`	Plots time scale vs entropy value.
`false`	No plot.

Example of a refined multiscale entropy curve for a normally distributed random number sequence using sample entropy over 5 time scales.



Outputs

`MSx` Refined multiscale entropy estimate at each time scale (τ), a vector of length **`Scales`**.

`Ci` Complexity index (area under the multiscale entropy curve), a scalar.

References

[48] [49] [50] [63] [64]

3.4.5 hMSEn: Hierarchical Multiscale Entropy

Syntax

```
[MSx, Sn, Ci] = hMSEn(Sig, Mobj, 'Scales', 3, 'RadNew', 0, 'Plotx', false)
MSx, Sn, Ci = hMSEn(Sig, Mobj, Scales = 3, RadNew = 0, Plotx = False)
MSx, Sn, Ci = hMSEn(Sig, Mobj, Scales = 3, RadNew = 0, Plotx = false)
```

Arguments

Sig Time series signal, a vector of length > 10.
The length of **Sig** (K) is halved at each scale. Only use the first 2^N data points are used such that $\min_N(K - 2^N)$.
i.e. For a signal of 5000 points, only the first 4096 points are used. For a signal of 1500 points, only the first 1024 points are used.

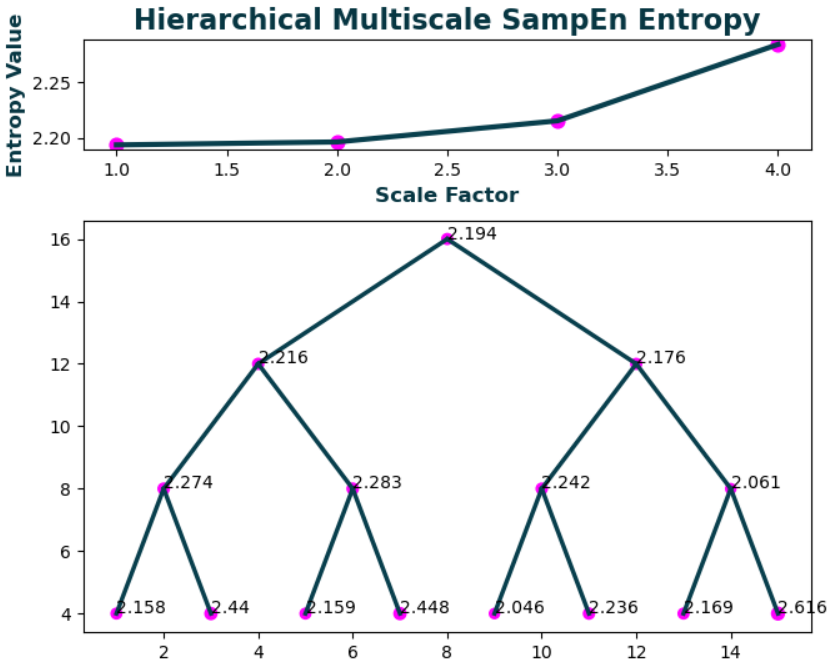
Mobj Multiscale Entropy object created with **MSobject()** - 3.4.1

Scales Number of time scales, an integer > 0.

RadNew Radius rescaling method, an integer in the range [0 4].
When the **Base** entropy method specified by **Mobj** is **SampEn** or **ApEn**, **RadNew** allows the radius threshold to be updated based on the grained signal at each time scale (X_τ). If a radius threshold value **(r)** is specified in **Mobj**, this becomes the rescaling coefficient, otherwise it is set to 0.2 (default). The value of **RadNew** specifies one of the following methods:

0	no rescaling
1	Standard Deviation - $r\sigma_{X_\tau}$
2	Variance - $r\sigma^2_{X_\tau}$
3	Mean Absolute Deviation - $r(\frac{1}{N}\sum\|X_\tau - \bar{X_\tau}\|)$
4	Median Absolute Deviation - $r(median(\|X_\tau - median(X_\tau)\|))$

Plotx A plot of the multiscale entropy curve
`true` Plots a curve of the average entropy value at each time scale (i.e. the multiscale entropy curve) and a hierarchical graph showing the entropy value of each node in the hierarchical tree decomposition.
`false` No plot.
Example of a multiscale entropy curve and a hierarchical tree graph for a normally distributed random number sequence using sample entropy over 4 time scales.



Outputs

MSx	Entropy estimate at each node in the hierarchical tree, a vector of length $2^{Scales} - 1$.
Sn	Average entropy value across each scale of hierarchical tree, a vector of length **Scales**.
Ci	Complexity index (area under the multiscale entropy curve), a scalar.

References

[65]

3.5 Multiscale Cross-Entropy Functions

Just as one can calculate multiscale entropy using any `Base` entropy, the same functionality is possible with multiscale cross-entropy using any `Cross`-entropy function (**`XApEn, XSampEn, XK2En, XCondEn, XPermEn, XSpecEn, XDistEn, XFuzzEn`**). To do so, we again use the **`MSobject`** function to pass a multiscale object (**`Mobj`**) to the multiscale cross-entropy functions.

Multiscale cross-entropy functions have three positional arguments:

the first data sequence **`Sig1`** (a vector),
the second data sequence **`Sig2`** (a vector),
and the multiscale entropy object, **`Mobj`** - 3.4.1.

****Note:** - *In versions previous to v1.0, only signals of the same length could be analysed with cross-entropy methods by passing a single Nx2 matrix. This is no longer the case - you must pass two individual signals to multiscale cross functions.*

Examples (shown in Julia syntax):

Original multiscale cross-entropy [48]

```
Mobj = MSobject(XSampEn)
xmse = XMSEn(Sig, Mobj)
```

Multiscale cross-distribution entropy using Rice's binning method and signal graining with empirical mode decomposition [53] [19]

```
Mobj = MSobject(XDistEn, Bins = "rice")
xmse = XMSEn(Sig, Mobj, Methodx = "imf")
```

Composite multiscale cross-conditional entropy with a 10-symbol data sequence, calculated up to 5 temporal scales [60] [18]

```
Mobj = MSobject(XCondEn, m = 5, c = 10)
cxmse = cXMSEn(Sig, Mobj, scales = 5)
```

Refined-Composite multiscale cross-entropy calculated in dits [61]

```
Mobj = MSobject(XSampEn, Logx = 10)
rcxmse = cXMSEn(Sig, Mobj, Refined = true)
```

Refined multiscale cross-permutation entropy calculated using an embedding dimension of 4 and a time delay of 4

```
Mobj = MSobject(XPermEn, m = 4, tau = 4)
rxmse = rXMSEn(Sig, Mobj)
```

3.5.1 MSobject: Multiscale Entropy Object

Syntax

```
Mobj = MSobject(EnType, varargin)
Mobj = MSobject(EnType, **kwargs)
Mobj = MSobject(EnType::Function, kwargs...)
```

Arguments

EnType — In **MatLab** and **Python**, **EnType** is a case-sensitive string corresponding to a valid **Base** or **Cross-** entropy function, e.g.'SyDyEn' or 'XDistEn', etc.

In **Julia**, **EnType** is a **Base** or **Cross-** entropy Function object, e.g. EntropyHub.XApEn (or XApEn if imported independently) etc.

varargin
****kwargs**
kwargs... — Any valid keyword arguments (Name/Value pairs) for the entropy function specified by **EnType**

Outputs

Mobj — Multiscale Entropy object.

3.5.2 XMSEn: Multiscale Cross-Entropy

Syntax

```
[MSx, Ci] = XMSEn(Sig1, Sig2, Mobj, 'Scales', 3, 'Methodx', 'coarse', 'RadNew',
0, 'Plotx', false)
MSx, Ci = XMSEn(Sig1, Sig2, Mobj, Scales = 3, Methodx = 'coarse', RadNew = 0,
Plotx = False)
MSx, Ci = XMSEn(Sig1, Sig2, Mobj, Scales = 3, Methodx = "coarse", RadNew = 0,
Plotx = false)
```

Arguments

Sig1 First data sequence, a vector > 10 elements.
Sig2 Second data sequence, a vector > 10 elements.
Mobj Multiscale Entropy object created with **MSobject()** - 3.4.1
Scales Number of grained time scales, a positive integer.
Methodx Type of graining method, one of the following strings:

"coarse" [48]

$$y_j^{(\tau)} = \frac{1}{\tau} \sum_{i=(j-1)\tau+1}^{j\tau} x_i, \qquad 1 \le j \le \frac{N}{\tau}$$

"generalized" [59]

$$y_j^{(\tau)} = \frac{1}{\tau} \sum_{k=0}^{\tau-1} (x_{j-k} - \bar{x})^2, \qquad 1 \le j \le N - \tau + 1$$

"modified"

$$y_j^{(\tau)} = \frac{1}{\tau} \sum_{k=0}^{\tau-1} x_{j-k}, \qquad 1 \le j \le N - \tau + 1$$

"timeshift" [57]

$$y_\beta^\tau = (x_\beta, x_{\beta+\tau}, x_{\beta+2\tau}, ..., x_{\beta+\lfloor \frac{N-\beta}{\tau} \rfloor \tau}) \qquad for \quad \beta = 1, 2, ..., \tau$$

$$TSME_\tau = \frac{1}{\tau} \sum_{\beta=1}^{\tau} F_{EnType}(y_\beta^\tau)$$

"imf" Grained time series at scale τ is the cumulative sum of intrinsic mode functions (IMF^1 to IMF^τ), where IMF^1 is the first sifting. [53]
***Note:** The empirical mode decomposition method use to derive the IMFs differs slightly between **MatLab, Python** and **Julia**, so **MSx** values will be inconsistent between the platforms.
****Note:** **Julia's** empirical mode decomposition method is unstable and may fully decompose highly stochastic or aperiodic signals.

RadNew Radius rescaling method, an integer in the range [0 4].
When the **Cross**-entropy method specified by **Mobj** is **XSampEn** or **XApEn**, **RadNew** allows the radius threshold to be updated based on the grained signal at each time scale (X_τ). If a radius threshold value **(r)** is specified in **Mobj**, this becomes the rescaling coefficient, otherwise it is set to 0.2 (default). The value of **RadNew** specifies one of the following methods:

0 no rescaling
1 Pooled Standard Deviation - $r\sigma_{X_\tau Y_\tau}$
2 Pooled Variance - $r\sigma^2_{X_\tau Y_\tau}$

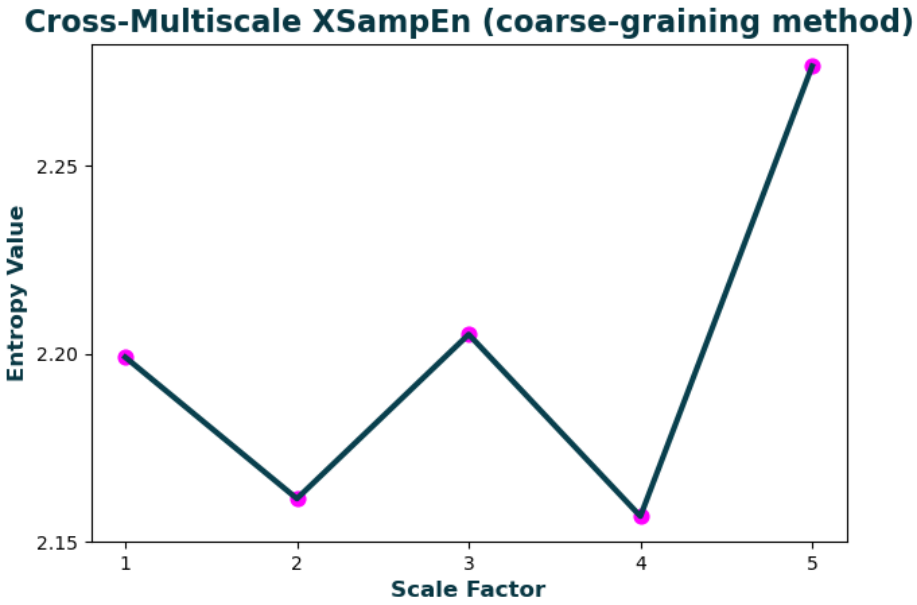
3	Combined Mean Absolute Deviation - $r(\frac{1}{N_{XY}} \sum \lvert XY_\tau - \bar{X}\bar{Y}_\tau \rvert)$, where XY is the concatenation of X and Y.
4	Combined Median Absolute Deviation - $r(median(\lvert XY_\tau - median(XY_\tau) \rvert))$ where XY is the concatenation of X and Y.

Plotx A plot of the multiscale entropy curve

`true` Plots time scale vs cross-entropy value.

`false` No plot.

An example multiscale cross-entropy curve of two normally-distributed random number sequences using cross-sample entropy over 5 coarse-grained time scales.



Outputs

MSx Multiscale cross-entropy estimate at each time scale (τ), a vector of length **Scales**.

Ci Complexity index (area under the multiscale entropy curve), a scalar.

References [48] [49] [50] [66] [67] [68] [69]

3.5.3 cXMSEn: Composite & Refined-Composite Multiscale Cross-Entropy

Syntax

```
[MSx, Ci] = cXMSEn(Sig1, Sig2, Mobj, 'Scales', 3, 'RadNew', 0, 'Refined', false,
'Plotx', false)
MSx, Ci = cXMSEn(Sig1, Sig2, Mobj, Scales = 3, RadNew = 0, Refined = False, Plotx
= False)
MSx, Ci = cXMSEn(Sig1, Sig2, Mobj, Scales = 3, RadNew = 0, Refined = false, Plotx
= false)
```

Arguments

Sig1 First data sequence, a vector > 10 elements.

Sig2 Second data sequence, a vector > 10 elements.

Mobj Multiscale Entropy object created with **MSobject()** - 3.4.1

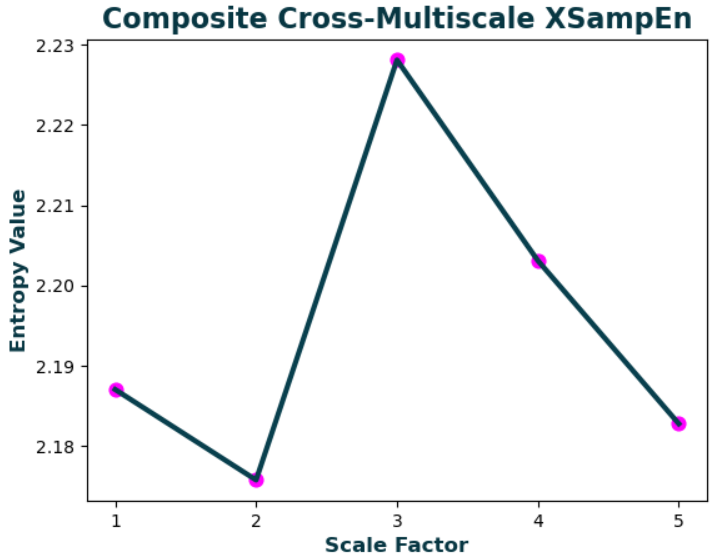
Scales Number of time scales, a positive integer.

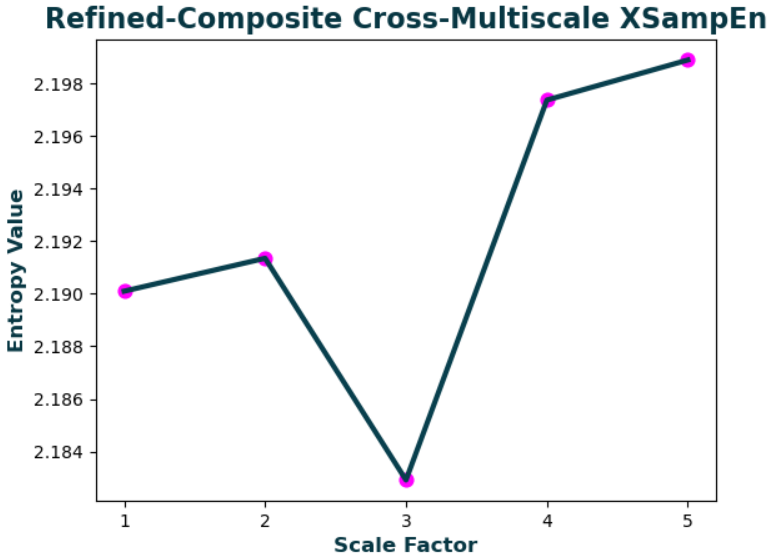
RadNew Radius rescaling method, an integer in the range [0 4].
When the **Cross**-entropy method specified by **Mobj** is **XSampEn** or **XApEn**, **RadNew** allows the radius threshold to be updated based on the grained signal at each time scale (X_τ). If a radius threshold value **(r)** is specified in **Mobj**, this becomes the rescaling coefficient, otherwise it is set to 0.2 (default). The value of **RadNew** specifies one of the following methods:

- **0** no rescaling
- **1** Pooled Standard Deviation - $r\sigma_{X_\tau Y_\tau}$
- **2** Pooled Variance - $r\sigma^2_{X_\tau Y_\tau}$
- **3** Combined Mean Absolute Deviation - $r(\frac{1}{N_{XY}}\sum|XY_\tau - \bar{X}\bar{Y}_\tau|)$, where XY is the concatenation of X and Y.
- **4** Combined Median Absolute Deviation - $r(median(|XY_\tau - median(XY_\tau)|))$ where XY is the concatenation of X and Y.

Refined When **Refined == true** and the entropy function (**EnType**) contained in **Mobj** is either **XSampEn** or **FuzzEn**, **cXMSEn** returns the refined-composite multiscale cross-entropy (**rcXMSEn**). [61] [62]

Plotx A plot of the multiscale entropy curve
`true` Plots time scale vs entropy value.
`false` No plot.
Example of composite multiscale cross-entropy and refined-composite multiscale cross-entropy curves for two sets of normally-distributed random number sequences using cross-sample entropy over 5 time scales.





Note: refined-composite multiscale sample entropy uses moving averaging to perform graining (analogous to the `modified` graining method in **`XMSEn`**), but refined-composite multiscale fuzzy entropy uses the moving variance (analogous to the `generalized` graining method in **`XMSEn`**).

Outputs

`MSx`	Composite multiscale cross-entropy estimate at each time scale (τ), a vector of length **`Scales`**.
`Ci`	Complexity index (area under the multiscale entropy curve), a scalar.

References [66] [67] [68] [69] [60] [62]

3.5.4 rXMSEn: Refined Multiscale Cross-Entropy

Syntax

```
[MSx, Ci] = rXMSEn(Sig1, Sig2, Mobj, 'Scales', 3, 'F_Order', 6, 'F_Num', 0.5,
'RadNew', 0, 'Plotx', false)
MSx, Ci = rXMSEn(Sig1, Sig2, Mobj, Scales = 3, F_Order = 6, F_Num = 0.5, RadNew
= 0, Plotx = False)
MSx, Ci = rXMSEn(Sig1, Sig2, Mobj, Scales = 3, F_Order = 6, F_Num = 0.5, RadNew
= 0, Plotx = false)
```

Arguments

Sig1 First data sequence, a vector > 10 elements.

Sig2 Second data sequence, a vector > 10 elements.

Mobj Multiscale Entropy object created with **MSobject()** - 3.4.1

Scales Number of time scales, a positive integer.

F_Order Butterworth low-pass filter order, a positive integer > 1, (default: 6)

F_Num Numerator of Butterworth low-pass filter cutoff frequency, where $[0 < F_{Num} < 1]$.
The cutoff frequency at each scale (τ) becomes: $F_c = \frac{F_{Num}}{\tau}$ (default: 0.5)

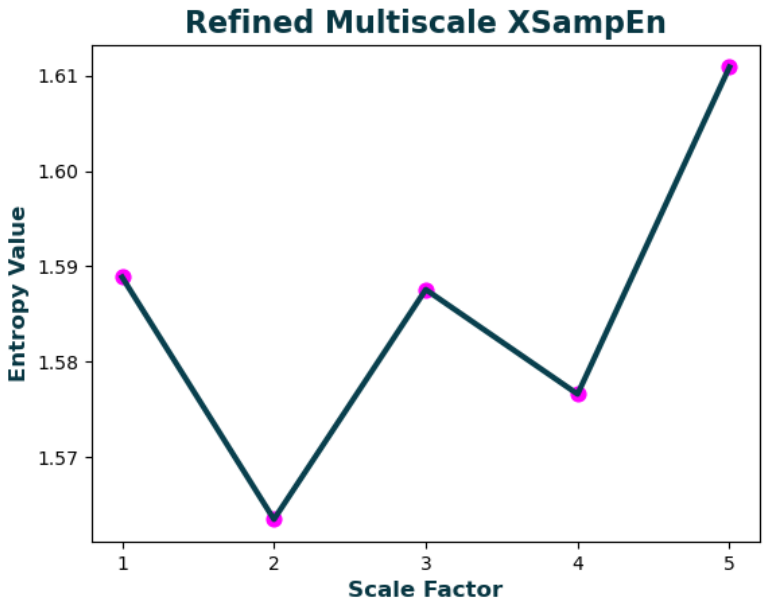
RadNew Radius rescaling method, an integer in the range [0 4].
When the **Cross**-entropy method specified by **Mobj** is **XSampEn** or **XApEn**, **RadNew** allows the radius threshold to be updated based on the grained signal at each time scale (X_τ). If a radius threshold value **(r)** is specified in **Mobj**, this becomes the rescaling coefficient, otherwise it is set to 0.2 (default). The value of **RadNew** specifies one of the following methods:

- **0** no rescaling
- **1** Pooled Standard Deviation - $r\sigma_{X_\tau Y_\tau}$
- **2** Pooled Variance - $r\sigma^2_{X_\tau Y_\tau}$
- **3** Combined Mean Absolute Deviation - $r(\frac{1}{N_{XY}} \sum |XY_\tau - \bar{X}\bar{Y}_\tau|)$, where XY is the concatenation of X and Y.
- **4** Combined Median Absolute Deviation - $r(median(|XY_\tau - median(XY_\tau)|))$ where XY is the concatenation of X and Y.

Plotx A plot of the multiscale entropy curve

- `true` Plots time scale vs entropy value.
- `false` No plot.

Example of a refined multiscale cross-entropy curve for two normally distributed random number sequences using cross-sample entropy over 5 time scales.



Outputs

MSx Refined multiscale cross-entropy estimate at each time scale (τ), a vector of length **Scales**.

Ci Complexity index (area under the multiscale entropy curve), a scalar.

References [48] [66] [63]

3.5.5 hXMSEn: Hierarchical Multiscale Cross-Entropy

Syntax

```
[MSx, Ci] = hXMSEn(Sig1, Sig2, Mobj, 'Scales', 3, 'RadNew', 0, 'Plotx', false)
MSx, Ci = hXMSEn(Sig1, Sig2, Mobj, Scales = 3, RadNew = 0, Plotx = False)
MSx, Ci = hXMSEn(Sig1, Sig2, Mobj, Scales = 3, RadNew = 0, Plotx = false)
```

Arguments

Sig1 First data sequence, a vector > 10 elements.

Sig2 Second data sequence, a vector > 10 elements. The length of **Sig1 / Sig2** (K) is halved at each scale. Only use the first 2^N data points are used such that $\min_N(K-2^N)$. i.e. For signals of 5000 points, only the first 4096 points are used. For signals of 1500 points, only the first 1024 points are used.

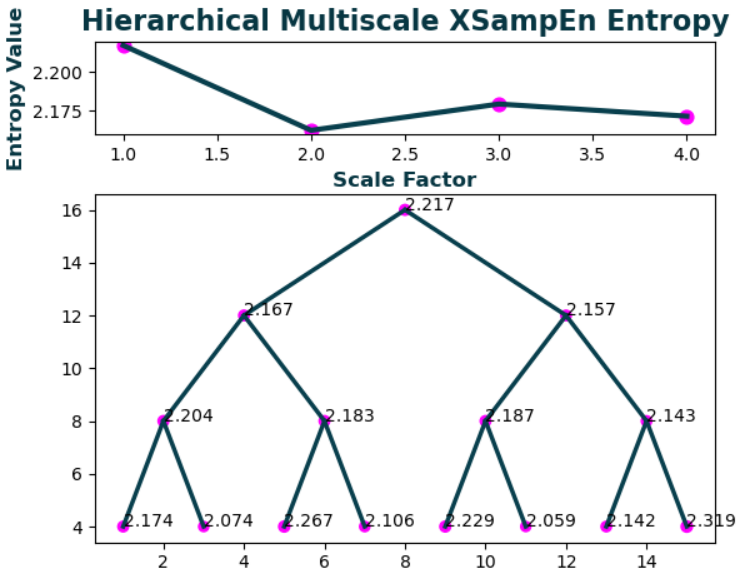
Mobj Multiscale Entropy object created with **MSobject()** - 3.4.1

Scales Number of time scales, a positive integer.

RadNew Radius rescaling method, an integer in the range [0 4].
When the **Cross**-entropy method specified by **Mobj** is **XSampEn** or **XApEn**, **RadNew** allows the radius threshold to be updated based on the grained signal at each time scale (X_τ). If a radius threshold value **(r)** is specified in **Mobj**, this becomes the rescaling coefficient, otherwise it is set to 0.2 (default). The value of **RadNew** specifies one of the following methods:

- **0** no rescaling
- **1** Pooled Standard Deviation - $r\sigma_{X_\tau Y_\tau}$
- **2** Pooled Variance - $r\sigma^2_{X_\tau Y_\tau}$
- **3** Combined Mean Absolute Deviation - $r(\frac{1}{N_{XY}}\sum|XY_\tau - \bar{X}\bar{Y}_\tau|)$, where XY is the concatenation of X and Y.
- **4** Combined Median Absolute Deviation - $r(median(|XY_\tau - median(XY_\tau)|))$ where XY is the concatenation of X and Y.

Plotx A plot of the multiscale entropy curve
`true` Plots a curve of the average cross-entropy value at each time scale (i.e. the multiscale entropy curve) and a hierarchical graph showing the cross-entropy value of each node in the hierarchical tree decomposition.
`false` No plot.
Example of a multiscale cross-entropy curve and a hierarchical tree graph for two normally distributed random number sequences using cross-sample entropy over 4 time scales.



Outputs

MSx Cross-entropy estimate at each node in the hierarchical tree, a vector of length $2^{Scales} - 1$.

Sn Average cross-entropy value across each scale of hierarchical tree, a vector of length **Scales**.

Ci Complexity index (area under the multiscale entropy curve), a scalar.

References [65]

3.6 Multivariate Multiscale Entropy Functions

Just as one can calculate multiscale entropy using any `Base / Cross-` entropy, the same functionality is possible with *multivariate* multiscale entropy using any *multivariate* entropy function (**`MvSampEn, MvDispEn, MvCoSiEn, MvPermEn, MvFuzzEn`**). To do so, we again use the **`MSobject`** function to pass a multiscale object (**`Mobj`**) to the multivariate multiscale entropy functions.

> Multivariate multiscale entropy functions have 2 positional arguments:
>
> the multivariate dataset **`Data`** (a NxM matrix, where N> 1, M> 1),
> and the multiscale entropy object, **`Mobj`** - 3.4.1.

Examples (shown in Julia syntax):

Original multivariate multiscale entropy [79]

```
Mobj = MSobject(MvSampEn)
mvmse = MvMSEn(Data, Mobj)
```

Generalized multivariate multiscale dispersion entropy using the k-means symbolic sequence transform and the "full" method for m+1 subspace extraction [79] [82]

```
Mobj = MSobject(MvDisEn, Typex = "kmeans", Methodx = "v2")
mvmse = MvMSEn(Data, Mobj, Methodx = "generalized")
```

Refined-Composite multivariate multiscale fuzzy entropy calculated in dits using a triangular fuzzy function (r = 3) and with embedding dimensions (assuming a 5-sequence multivariate dataset) m = [2,3,1,3,4] [81]

```
Mobj = MSobject(MvFuzzEn, Logx = 10, Fx = "triangular", r = 3, m = [2,3,1,3,4])
rcmvmse = cMvMSEn(Data, Mobj, Refined = true)
```

3.6.1 MvMSEn: Multivariate Multiscale Entropy

Syntax

```
[MSx, Ci] = MvMSEn(Data, Mobj, 'Scales', 3, 'Methodx', 'coarse', 'Plotx', false)
MSx, Ci = MvMSEn(Data, Mobj, Scales = 3, Methodx = 'coarse', Plotx = False)
MSx, Ci = MvMSEn(Data, Mobj, Scales = 3, Methodx = "coarse", Plotx = false)
```

ATTENTION

By default, the **MvSampEn** and **MvFuzzEn** multivariate entropy algorithms estimate entropy values using the "full" method by comparing delay vectors across all possible m+1 expansions of the embedding space as applied in [79] [80] [81]. These methods are **not lower-bounded to 0**, like most entropy algorithms, so **MvMSEn** may return negative entropy values if the base multivariate entropy function is **MvSampEn** and **MvFuzzEn**, even for stochastic processes...

Arguments

Data Multivariate dataset, NxM matrix of N (> 10) observations (rows) and M (> 1) univariate data sequences (cols).

Mobj Multiscale Entropy object created with **MSobject()** - 3.4.1

Scales Number of grained time scales, a positive integer.

Methodx Type of graining method, one of the following strings:

"coarse" [48]

$$y_j^{(\tau)} = \frac{1}{\tau} \sum_{i=(j-1)\tau+1}^{j\tau} x_i, \qquad 1 \le j \le \frac{N}{\tau}$$

"generalized" [59]

$$y_j^{(\tau)} = \frac{1}{\tau} \sum_{k=0}^{\tau-1} (x_{j-k} - \bar{x})^2, \qquad 1 \le j \le N - \tau + 1$$

"modified"

$$y_j^{(\tau)} = \frac{1}{\tau} \sum_{k=0}^{\tau-1} x_{j-k}, \qquad 1 \le j \le N - \tau + 1$$

Plotx A plot of the multiscale entropy curve

`true` Plots time scale vs entropy value.

`false` No plot.

Outputs

MSx Multivariate multiscale entropy estimate at each time scale (τ), a vector of length **Scales**.

Ci Complexity index (area under the multiscale entropy curve), a scalar.

References [48] [49] [79] [80]

3.6.2 cMvMSEn: Composite & Refined-Composite Multivariate Multiscale Entropy

Syntax

```
[MSx, Ci] = cMvMSEn(Data, Mobj, 'Scales', 3, 'Refined', false, 'Plotx', false)
MSx, Ci = cMvMSEn(Data, Mobj, Scales = 3, Refined = False, Plotx = False)
MSx, Ci = cMvMSEn(Data, Mobj, Scales = 3, Refined = false, Plotx = false)
```

ATTENTION

By default, the **MvSampEn** and **MvFuzzEn** multivariate entropy algorithms estimate entropy values using the "full" method by comparing delay vectors across all possible m+1 expansions of the embedding space as applied in [79] [80] [81]. These methods are **not lower-bounded to 0**, like most entropy algorithms, so **MvMSEn** may return negative entropy values if the base multivariate entropy function is **MvSampEn** and **MvFuzzEn**, even for stochastic processes...

Arguments

Data	Multivariate dataset, NxM matrix of N (> 10) observations (rows) and M (> 1) univariate data sequences (cols).
Mobj	Multiscale Entropy object created with **MSobject()** - 3.4.1
Scales	Number of time scales, a positive integer.
Refined	When **Refined == true** and the entropy function (**EnType**) contained in **Mobj** is either **MvSampEn**, or **MvFuzzEn**, **cMvMSEn** returns the refined-composite multivariate multiscale entropy (**rcMvMSEn**). [79] [81]
Plotx	A plot of the multiscale entropy curve `true` Plots time scale vs entropy value. `false` No plot.

Note: refined-composite multivariate multiscale sample entropy uses moving averaging to perform graining (analogous to the `modified` graining method in **MvMSEn**), but refined-composite multivariate multiscale fuzzy entropy uses the moving standard deviation (analogous to the `generalized` graining method in **MvMSEn**).

Outputs

MSx	Composite / refined-composite multivariate multiscale entropy estimate at each time scale (τ), a vector of length **Scales**.
Ci	Complexity index (area under the multiscale entropy curve), a scalar.

References [79] [81]

3.7 Bidimensional Entropy Functions

While EntropyHub functions primarily apply to time series data, with the following bidimensional entropy functions one can estimate the entropy of two-dimensional (2D) matrices. Hence, bidimensional entropy functions are useful for applications such as image analysis.

IMPORTANT NOTE

Each bidimensional entropy function (**SampEn2D, FuzzEn2D, DistEn2D, DispEn2D, PermEn2D, EspEn2D**) has an important keyword argument - **Lock**. Bidimensional entropy functions are "locked" by default (**Lock** == `true`) to only permit matrices with a maximum size of 128 x 128.

The reason for this is because there are hundreds of millions of pairwise calculations performed in the estimation of bidimensional entropy, so memory errors often occur when storing data on RAM.
e.g. For a matrix of size [200 x 200], an embedding dimension (**m**) = 3, and a time delay (**tau**) = 1, there are 753,049,836 pairwise matrix comparisons (6,777,448,524 elemental subtractions).

To pass matrices with sizes greater than [128 x 128], set **Lock** = `false`.

*** WARNING: unlocking the permitted matrix size may cause your programming environment to crash.***

3.7.1 SampEn2D: Bidimensional Sample Entropy

Syntax

```
[SE2D, Phi1, Phi2] = SampEn2D(Mat, 'm', floor(size(Mat)/10), 'tau', 1, 'r',
0.2*std(Mat), 'Logx', exp(1), 'Lock', true)
SE2D, Phi1, Phi2 = SampEn2D(Mat, m = Mat.shape//10, tau = 1, r = 0.2*np.std(Mat),
Logx = np.exp(1), Lock = True)
SE2D, Phi1, Phi2 = SampEn2D(Mat, m = floor(Int,size(Mat)./10), tau = 1,
r = 0.2*std(Mat), Logx = exp(1)), Lock = true)
```

Arguments

Mat	N x M matrix, where N, M > 10.
m	Embedding dimension, [default: (floor(N/10) floor(M/10))] - an integer > 1 for square submatrix embedding, or - a two-element tuple of integers > 1 representing the height and width of the template submatrix.
tau	Time delay, a positive integer.
r	Distance threshold value, a positive scalar.
Logx	Logarithm base in the entropy formula, a positive scalar.
Lock	See note on matrix size locking `true` Matrix height (N) and width (M) must be < 128 elements. `false` Matrix of any size can be passed.

Outputs

SE2D	Bidimensional sample entropy estimate.
Phi1	The number of matched submatrices for embedding dimensions (**m**).
Phi2	The number of matched submatrices for embedding dimensions (**m+1**).

References

[70]

3.7.2 FuzzEn2D: Bidimensional Fuzzy Entropy

Syntax

```
Fuzz2D = FuzzEn2D(Mat, 'm', floor(size(Mat)/10), 'tau', 1, 'Fx', 'default', 'r',
[0.2*std(Mat(:)), 2], 'Logx', exp(1), 'Lock', true)
Fuzz2D = FuzzEn2D(Mat, m = Mat.shape//10, tau = 1, Fx = 'default', r = (0.2*np.std(Mat[:
2), Logx = np.exp(1), Lock = True)
Fuzz2D = FuzzEn2D(Mat, m = floor(Int, size(Mat)./10), tau = 1, Fx = "default",
r = (0.2*std(Mat[:]), 2), Logx = exp(1), Lock = true)
```

Arguments

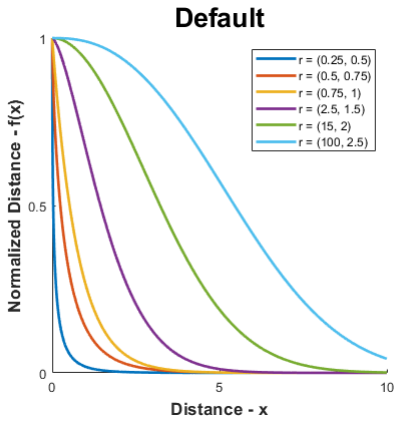
Mat N x M matrix, where N, M > 10.

m Embedding dimension, [default: (floor(N/10) floor(M/10))]
- an integer > 1 for square submatrix embedding, or
- a two-element tuple of integers > 1 representing the height and width of the template submatrix.

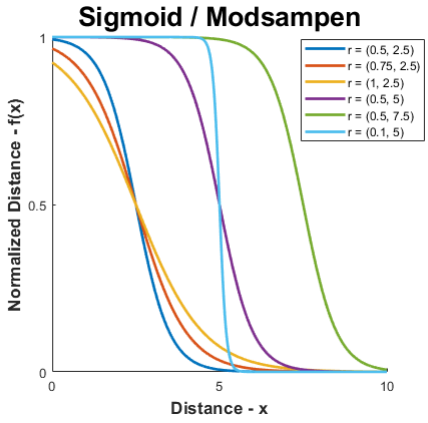
tau Time delay, a positive integer.

Fx Type of fuzzy function for distance transformation, one of the following strings:

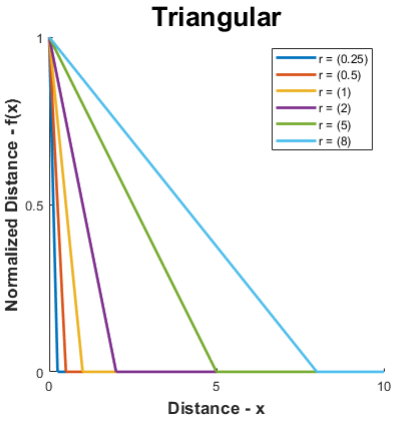
"default" $f(x) = exp(-\frac{x^{r_2}}{r_1})$



"sigmoid"/"modsampen" $f(x) = (1 + exp(\frac{x-r_2}{r_1}))^{-1}$

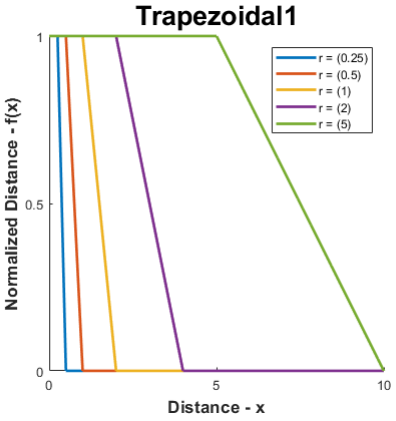


"triangular" $f(x) = \begin{cases} 1 - \frac{x}{r}, & x \leq r \\ 0, & x > r \end{cases}$



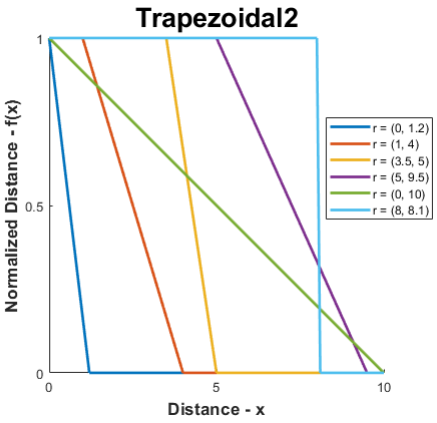
"trapezoidal1"

$$f(x) = \begin{cases} 1, & x < r \\ 2 - \frac{x}{r}, & r \leq x < 2r \\ 0, & x \geq 2r \end{cases}$$



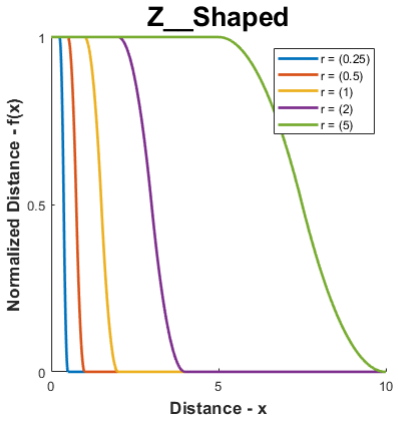
"trapezoidal2"

$$f(x) = \begin{cases} 1, & x < r \\ 1 - \frac{x - r_{min}}{r_{max} - r_{min}}, & r \leq x < r_{max} \\ 0, & x \geq r_{max} \end{cases}$$

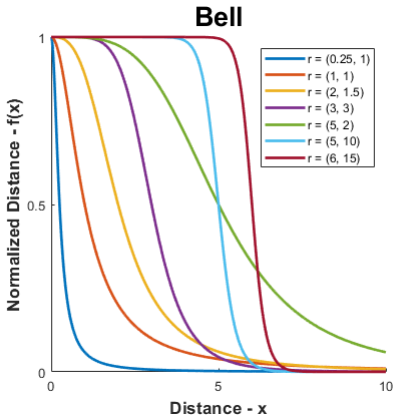


"z_shaped"

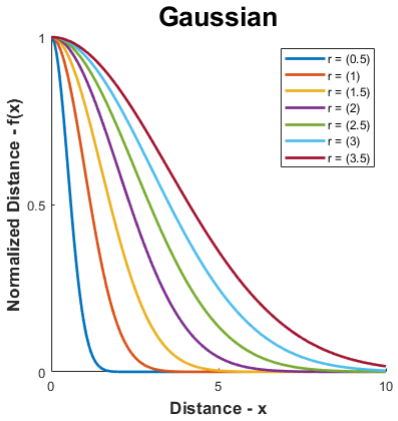
$$f(x) = \begin{cases} 1, & x < r \\ 1 - 2\left(\frac{x-r}{r}\right)^2, & r < x < \frac{3}{2}r \\ 2\left(\frac{x-r}{r}\right)^2, & \frac{3}{2}r < x < 2r \\ 0, & x > 2r \end{cases}$$



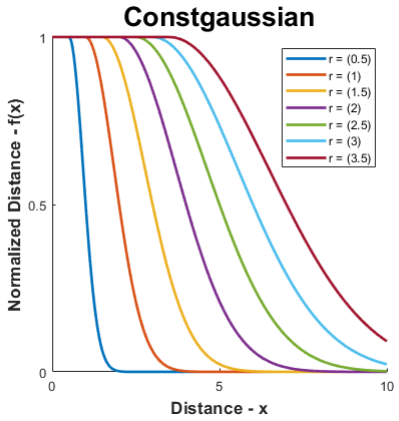
"bell" $f(x) = \frac{1}{1+\left|\frac{x}{r_1}\right|^{r_2}}$



"gaussian" $f(x) = exp(-\frac{x^2}{2r^2})$

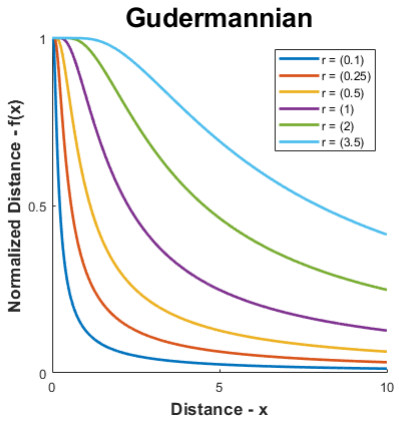


"constgaussian" $f(x) = \begin{cases} 1, & x < r_{min} \\ exp(-\log(2)\frac{(x-r)^2}{r}), & x \geq r_{max} \end{cases}$



"gudermannian"

$g(x) = atan(\frac{tanh(r_1)}{x})$
$f(x) = \frac{g(x)}{g(x_{max})}$
Note: Distances are normalized w.r.t. maximum distance relative to each state vector.



r Parameters of the fuzzy function specified by **Fx**, a 1 element scalar or a 2 element tuple of positive values depending on the fuzzy function as shown above.

Default	Two element tuple (or vector in MatLab)
Sigmoid/ModSampEn	Two element tuple (or vector in MatLab)
Trapezoidal2	Two element tuple (or vector in MatLab)
Bell	Two element tuple (or vector in MatLab)
Trapezoidal1	A scalar value
Triangular	A scalar value
Z_Shaped	A scalar value
Gaussian	A scalar value
ConstGaussian	A scalar value
Gudermannian	A scalar value

Logx Logarithm base in the entropy formula, a positive scalar.

Lock See note on matrix size locking

`true`	Matrix height (N) and width (M) must be < 128 elements.
`false`	Matrix of any size can be passed.

Outputs

Fuzz2D Bidimensional fuzzy entropy estimate.

References

[71], [72]

3.7.3 DistEn2D: Bidimensional Distribution Entropy

Syntax

```
Dist2D = DistEn2D(Mat, 'm', floor(size(Mat)/10), 'tau', 1, 'Bins', 'sturges',
'Logx', 2, 'Norm', true, 'Lock', true)
Dist2D = DistEn2D(Mat, m = Mat.shape//10, tau = 1, Bins = 'sturges', Logx = 2,
Norm = True, Lock = True)
Dist2D = DistEn2D(Mat, m = floor(Int, size(Mat)./10), tau = 1, Bins = "Sturges",
Logx = 2, Norm = true Lock = true)
```

Arguments

Mat	N x M matrix, where N, M > 10.
m	Embedding dimension, [default: (floor(N/10) floor(M/10))] - an integer > 1 for square submatrix embedding, or - a two-element tuple of integers > 1 representing the height and width of the template submatrix.
tau	Time delay, a positive integer.
Bins	Histogram binning method, in integer > 1 indicating the number of bins, or one of the following strings: "sturges", "sqrt", "rice", "doanes" [default: "sturges"]
Logx	Logarithm base in the entropy formula, a positive scalar. (Enter 0 for natural logarithm)
Norm	Normalization of **Dist2D** value: `false` no normalisation `true` normalises w.r.t number of histogram bins (default)
Lock	See note on matrix size locking `true` Matrix height (N) and width (M) must be < 128 elements. `false` Matrix of any size can be passed.

Outputs

Dist2D	Bidimensional distribution entropy estimate.

References

[73],

3.7.4 DispEn2D: Bidimensional Dispersion Entropy

Syntax

```
[Disp2D, RDE] = DispEn2D(Mat, 'm', floor(size(Mat)/10), 'tau', 1, 'c', 3, 'Typex',
'ncdf', 'Logx', exp(1), 'Norm', false, 'Lock', true)
Disp2D, RDE = DispEn2D(Mat, m = Mat.shape//10, tau = 1, c = 3, Typex = 'ncdf',
Logx = np.exp(1), Norm = False, Lock = True)
Disp2D, RDE = DispEn2D(Mat, m = floor(Int, size(Mat)./10), tau = 1, c = 3, Typex
= "ncdf", Logx = exp(1), Norm = false, Lock = true)
```

Arguments

Mat N x M matrix, where N, M > 10.

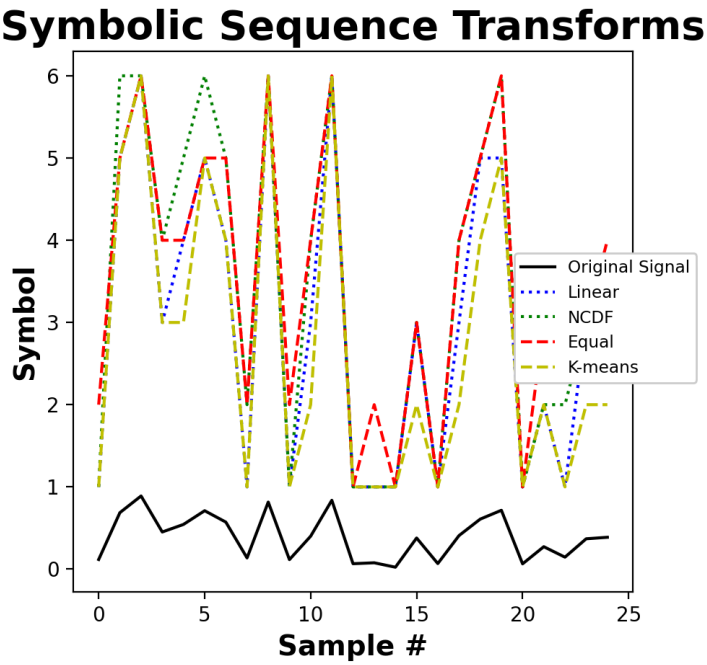
m Embedding dimension, [default: (floor(N/10) floor(M/10))]
- an integer > 1 for square submatrix embedding, or
- a two-element tuple of integers > 1 representing the height and width of the template submatrix.

tau Time delay, a positive integer.

c Number of symbols in transform, an integer > 1.

Typex Type of symbolic sequence transform, one of the following strings:

- **"ncdf"** Normalised cumulative distribution function [22]
- **"kmeans"** K-means clustering algorithm.
  **Note: The *"kmeans"* algorithm uses random initialization conditions. This causes results to vary slightly each time it is called.
- **"linear"** Linear segmentation of signal range
- **"finesort"** Fine-sorted dispersion entropy [25]
- **"equal"** Approx. equal number of symbols.



Logx Logarithm base in the entropy formula, a positive scalar.

Norm Normalization of **Disp2D** and **RDE** values:

- `false` no normalisation
- `true` normalises w.r.t number of possible dispersion patterns (default)

Lock See note on matrix size locking
- `true` Matrix height (N) and width (M) must be < 128 elements.
- `false` Matrix of any size can be passed.

Outputs

Dist2D Bidimensional dispersion entropy estimate.
RDE Bidimensional reverse dispersion entropy estimate.

References

[74],

3.7.5 PermEn2D: Bidimensional Permutation Entropy

Syntax

```
Perm2D = PermEn2D(Mat, 'm', floor(size(Mat)/10), 'tau', 1, 'Logx', exp(1), 'Norm', true, 'Lock', true)
Perm2D = PermEn2D(Mat, m = Mat.shape//10, tau = 1, Logx = np.exp(1), Norm = True, Lock = True)
Perm2D = PermEn2D(Mat, m = floor(Int, size(Mat)./10), tau = 1, Logx = exp(1), Norm = true, Lock = true)
```

IMPORTANT NOTE

The original bidimensional permutation entropy algorithms [75] [76] do not account for equal-valued elements of the embedding matrices. To overcome this, PermEn2D uses the lowest common rank for such instances. For example, given an embedding matrix A where,

$$\mathrm{A} = \begin{bmatrix} 3.4 & 5.5 & 7.3 \\ 2.1 & 6 & 9.9 \\ 7.3 & 1.1 & 2.1 \end{bmatrix}$$

would normally be mapped to an ordinal pattern like so,

$$\begin{bmatrix} 3.4 & 5.5 & 7.3 & 2.1 & 6 & 9.9 & 7.3 & 1.1 & 2.1 \end{bmatrix} =>$$
$$\begin{bmatrix} 8 & 4 & 9 & 1 & 2 & 5 & 3 & 7 & 6 \end{bmatrix}$$

However, indices 4 & 9, and 3 & 7 have the same values, 2.1 and 7.3 respectively. Instead, PermEn2D uses the ordinal pattern:

$$\begin{bmatrix} 8 & 4 & 4 & 1 & 2 & 5 & 3 & 3 & 6 \end{bmatrix}$$

where the lowest ranks (4 & 3) are used instead (of 9 & 7).
Therefore, the number of possible permutations is no longer $(m_x * m_y)!$, but $(m_x * m_y)^{m_x * m_y}$. Here, the **Perm2D** value is normalized by the maximum Shannon entropy ($S_{max} = \log((m_x * m_y)!)$ assuming that no equal values are found in the permutation motif matrices, as presented in [75].

Arguments

Mat	N x M matrix, where N, M > 10.
m	Embedding dimension, [default: (floor(N/10) floor(M/10))] - an integer > 1 for square submatrix embedding, or - a two-element tuple of integers > 1 representing the height and width of the template submatrix.
tau	Time delay, a positive integer.
Norm	Normalization of **Perm2D** value: `false` no normalisation `true` normalises w.r.t maximum Shannon Entropy ($S_{max} = \log((m_x * m_y)!)$) (default)

Logx Logarithm base in the entropy formula, a positive scalar.

Lock See note on matrix size locking

`true`	Matrix height (N) and width (M) must be < 128 elements.
`false`	Matrix of any size can be passed.

Outputs

Perm2D Bidimensional permutation entropy estimate.

References

[75],[76], [83]

3.7.6 EspEn2D: Bidimensional Espinosa Entropy

Syntax

```
Esp2D = EspEn2D(Mat, 'm', floor(size(Mat)/10), 'tau', 1, 'r', 20, 'ps', 0.7,
'Logx', exp(1), 'Lock', true)
Esp2D = EspEn2D(Mat, m = Mat.shape//10, tau = 1, r = 20, ps = 0.7, Logx = np.exp(1),
Lock = True)
Esp2D = EspEn2D(Mat, m = floor(Int,size(Mat)./10), tau = 1, r = 20, ps = 0.7,
Logx = exp(1)), Lock = true)
```

Arguments

Mat	N x M matrix, where N, M > 10.
m	Embedding dimension, [default: (floor(N/10) floor(M/10))] - an integer > 1 for square submatrix embedding, or - a two-element tuple of integers > 1 representing the height and width of the template submatrix.
tau	Time delay, a positive integer.
r	Tolerance threshold value, a positive scalar.
ps	Percentage similarity value, a value in range [0 1].
Logx	Logarithm base in the entropy formula, a positive scalar.
Lock	See note on matrix size locking `true` Matrix height (N) and width (M) must be < 128 elements. `false` Matrix of any size can be passed.

Outputs

Esp2D	Bidimensional Espinosa entropy estimate.

References [77]

3.8 Other Functions

3.8.1 ExampleData(): Import Example Datasets

Syntax

```
Data = ExampleData(Name)
Data = ExampleData(Name)
Data = ExampleData(Name)
```

Arguments

Name One of the following strings:

'uniform'	vector of uniformly distributed random numbers in range [0 1]
'gaussian'	vector of normally distributed random numbers with $\mu = 0, \sigma = 1$
'randintegers'	vector of uniformly distributed pseudorandom integers in range [1 8]
'chirp'	vector of chirp signal with the following parameters: $f_0 = .01, t_1 = 4000, f_1 = .025$
'lorenz'	3-column matrix: X, Y, Z components of the Lorenz system $(\sigma : 10, \beta : \frac{8}{3}, rho : 28), [X_o = 10, Y_o = 20, Z_o = 10]$
'henon'	2-column matrix: X, Y components of the Henon attractor $(\alpha : 1.4, \beta : .3), [X_o = 0, Y_o = 0]$
'uniform2'	2-column matrix: uniformly distributed random numbers in range [0 1]
'gaussian2'	2-column matrix: normally distributed random numbers with $\mu = 0, \sigma = 1$
'randintegers2'	2-column matrix: uniformly distributed pseudorandom integers in range [1 8]
'uniform_Mat'	Matrix of uniformly distributed random numbers in range [0 1]
'gaussian_Mat'	Matrix of normally distributed random numbers with $\mu = 0, \sigma = 1$
'randintegers_Mat'	Matrix of uniformly distributed pseudorandom integers in range [1 8]
'mandelbrot_Mat'	Matrix of image of fractal generated from the mandelbrot set
'entropyhub_Mat'	Matrix of image of the entropyhub logo

Outputs

Data The vector or matrix containing the desired dataset.

3.8.2 WindowData(): Data Windowing Tool

Syntax

```
WinData, Log = WindowData(Data, 'WinLen', Int32(floor(size(Data,1)/5)),
   'Overlap', 0, 'Mode', 'exclude')
WinData, Log = WindowData(Data, WinLen = Data.shape[0]//5,
   Overlap = 0, Mode = 'exclude')
WinData, Log = WindowData(Data, WinLen = floor(Int, size(Data,1)/5),
   Overlap = 0, Mode = "exclude")
```

Arguments

WinData	Number of elements in each window, a positive integer (> 10)
Overlap	Number of overlapping elements between windows, a positive integer ($< WinLen$)
Mode	Decision to include or exclude any remaining sequence elements ($< WinLen$) that do not fill the window, a string - either "include" or "exclude" (default).

Outputs

WinData	Collection of windowed subsequences. If Data is a univariate sequence (vector), Windata is a collection of vectors. If Data is a set of multivariate sequences (NxM matrix), each of M columns is treated as a sequence with N elements and WinData is a collection of matrices.
Log	Struct / dictionary containing information about the windowing process, including:

DataType	The type of data sequence passed as Data
DataLength	The number of sequence elements in Data
WindowLength	The number of elements in each window of WinData
WindowOverlap	The number of overlapping elements between windows
TotalWindows	The number of windows extracted from Data
Mode	Decision to include or exclude any remaining sequence elements ($< WinLen$) that do not fill the window.

4

Examples

The following sections provide some basic examples of EntropyHub functions. These examples are merely a snippet of the full range of EntropyHub functionality.

There is a custom documentation section installed with the toolkit in MatLab which provides several useful examples of every function in more detail than what is shown here. Thus, if you are using EntropyHub for MatLab, we recommend that you consult the custom EntropyHub documentation in MatLab for more in-depth examples.

In the following examples, signals / data are imported into MatLab/Python/Julia using the `ExampleData()- 3.8.1` function from EntropyHub.

NOTE

To use the **`ExampleData() - 3.8.1`** function as outlined in the examples below, **an internet connection is required.**

THINGS TO REMEMBER

Parameters of the base or cross- entropy methods are passed to multiscale and multiscale cross- functions using the multiscale entropy object using **`MSobject`** - **3.4.1**.
Base and cross- entropy methods are declared with `MSobject()` using a string name in **MatLab** and **Python**. In **Julia**, base and cross- entropy methods are passed as a **function**. See the MSobject examples (MatLab; 4.1.5, Python: 4.2.5, Julia: 4.3.5) in the following sections for more info.

Each bidimensional entropy function (**`SampEn2D`**, **`FuzzEn2D`**, **`DistEn2D`**, **`PermEn2D`**, **`EspEn2D`**) has an important keyword argument - **`Lock`**. Bidimensional entropy functions are "locked" by default (**`Lock`** == `true`) to only permit matrices with a maximum size of 128 x 128.

In hierarchical multiscale entropy (hMSEn) and hierarchical multiscale cross-entropy (hXMSEn) functions, the length of the time series signal(s) is halved at each scale. Thus, `hMSEn` and `hXMSEn` only use the first 2^N data points where $2^N <=$ the length of the original time series signal.
i.e. For a signal of 5000 points, only the first 4096 are used. For a signal of 1500 points, only the first 1024 are used.

4.1 MatLab:

4.1.1 Example 1: Sample Entropy

Import a signal of normally distributed random numbers $[\mu = 0, \sigma = 1]$, and calculate the sample entropy for each embedding dimension (m) from 0 to 4.

```
X = ExampleData('gaussian');

Samp = SampEn(X, 'm', 4)

>>> Samp = 1×5
    2.1789    2.1757    2.1820    2.2210    2.1756
```

Select the last value to get the sample entropy for m = 4.

```
Samp(end)
>>> ans = 2.1756
```

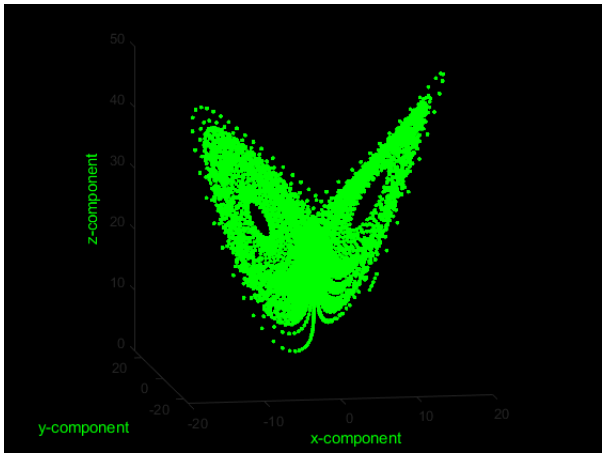
Calculate the sample entropy for each embedding dimension (m) from 0 to 4 with a time delay (tau) of 2 samples.

```

Samp = SampEn(X, 'm', 4, 'tau', 2)

>>> Samp = 1×5
    2.1789    2.1833    2.1880    2.1892    2.1441
```

4.1.2 Example 2: (Fine-Grained) Permutation Entropy

Import the x, y, and z components of the Lorenz system of equations.

```
Data = ExampleData('lorenz');
figure('Color', 'k')

plot3(Data(:,1), Data(:,2), Data(:,3), 'g.')
xlabel('x-component','color','g'),
ylabel('y-component','color','g'),
zlabel('z-component','color','g'),
view(-10,10), set(gca,'color','k'), axis square
```



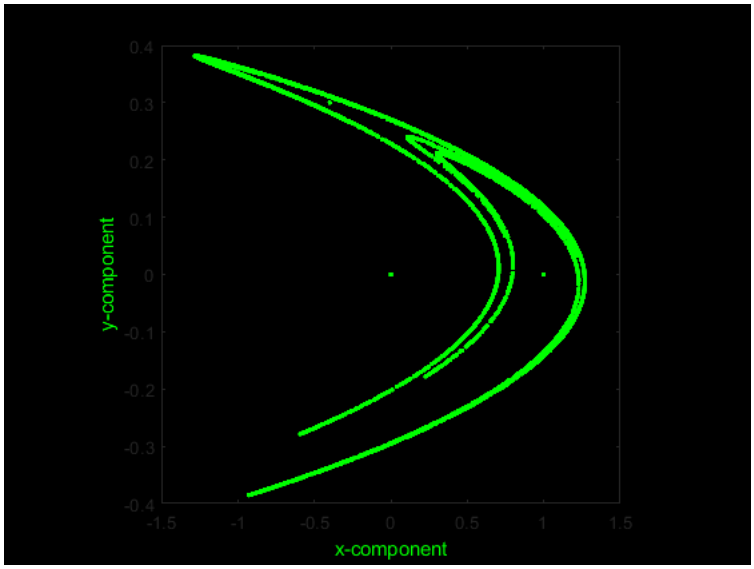
Calculate fine-grained permutation entropy of the z component in dits (logarithm base 10) with an embedding dimension of 3, time delay of 2, an alpha parameter of 1.234. Return `Pnorm` normalised w.r.t the number of all possible permutations (m!) and the condition permutation entropy (`cPE`) estimate.

```
Z = Data(:,3);
[Perm, Pnorm, cPE] = PermEn(Z, 'm', 3, 'tau', 2, 'Typex', ...
                'finegrain', 'tpx', 1.234, 'Logx', 10, ...
                'Norm', false)

>>> Perm = 1×3
       0    0.8687    0.9468
Pnorm = 1×3
       NaN    0.8687    0.4734
cPE = 1×2
           0.8687    0.0781
```

4.1.3 Example 3: Phase Entropy w/ Poincaré plot

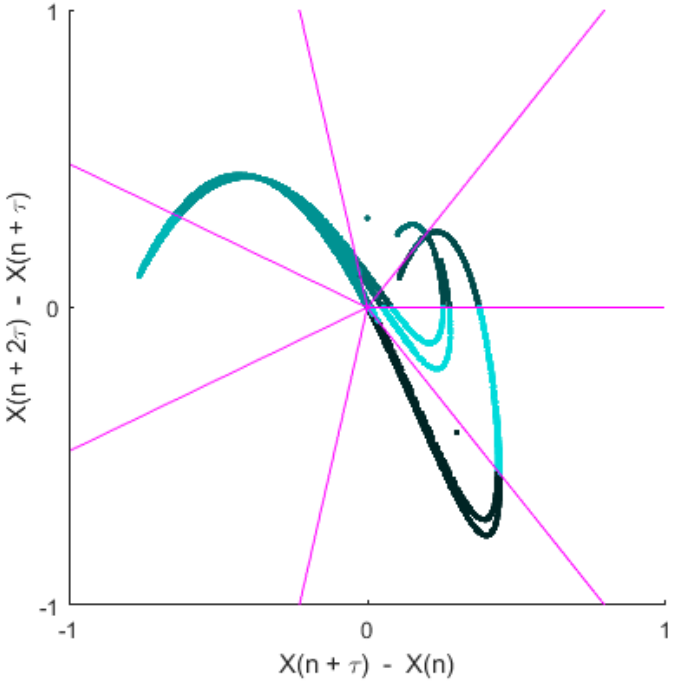
Import the x and y components of the Henon system of equations.

```
Data = ExampleData('henon');
figure('Color', 'k')
plot(Data(:,1), Data(:,2), 'g.')
xlabel('x-component','color','g'),
ylabel('y-component','color','g')
set(gca,'color','k'), axis square
```



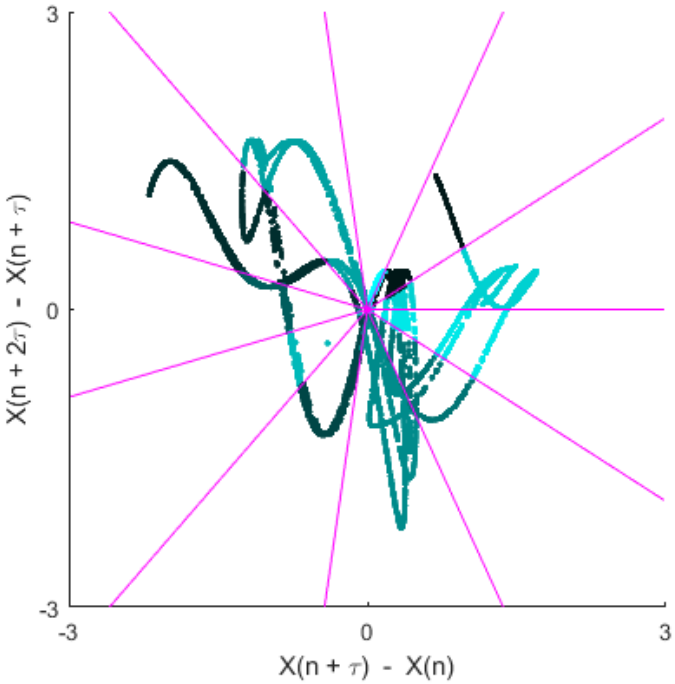
Calculate the phase entropy of the y-component in bits (logarithm base 2) without normalization using 7 angular partitions and return the second-order difference plot.

```
Y = Data(:,2);
Phas = PhasEn(Y, 'K', 7, 'Norm', false, 'Logx', 2, ...
        'Plotx', true)

>>> Phas = 2.0193
```



Calculate the phase entropy of the x-component using 11 angular partitions, a time delay of 2, and return the second-order difference plot.

```
X = Data(:,1);
Phas = PhasEn(X, 'K', 11, 'tau', 2, 'Plotx', true)

>>> Phas = 0.8395
```



4.1.4 Example 4: Cross-Distribution Entropy w/ Different Binning Methods

Import a signal of pseudorandom integers in the range [1, 8] and calculate the cross-distribution entropy with an embedding dimension of 5, a time delay (tau) of 3, and Sturges' bin selection method.

```
X = ExampleData('randintegers2');

XDist = XDistEn(X(:,1), X(:,2), 'm', 5, 'tau', 3)

>>> Note: 17/25 bins were empty
XDist = 0.5248
```

Use Rice's method to determine the number of histogram bins and return the probability of each bin (Ppi).

```
[XDist, Ppi] = XDistEn(X(:,1), X(:,2), 'm', 5, 'tau', 3, . . .
                'Bins', 'rice')

>>> Note: 407/415 bins were empty
        XDist = 0.2802
        Ppi = 1×8
    0.0000    0.0047    0.0368    0.1096    ...
            0.1978    0.2558    0.2421    0.1531
```

4.1.5 Example 5: Multiscale Entropy Object [MSobject()]

Create a multiscale entropy object (Mobj) for multiscale fuzzy entropy, calculated with an embedding dimension of 5, a time delay of 2, using a sigmoidal fuzzy function with the r scaling parameters (3, 1.2).

```
Mobj = MSobject('FuzzEn', 'm', 5, 'tau', 2, 'Fx', ...
   'sigmoid', 'r', [3, 1.2])

>>> Mobj = struct with fields:
    Func: @FuzzEn
       m: 5
     tau: 2
       r: [3 1.2000]
      Fx: 'sigmoid'
```

Create a multiscale entropy object (Mobj) for multiscale corrected-cross-conditional entropy, calculated with an embedding dimension of 6 and using a 11-symbolic data transform.

```
Mobj = MSobject('XCondEn', 'm', 6, 'c', 11)

>>> Mobj = struct with fields:
    Func: @XCondEn
       m: 6
       c: 11
```

4.1.6 Example 6: Multiscale [Increment] Entropy

Import a signal of uniformly distributed pseudorandom integers in the range [1,8] and create a multiscale entropy object with the following parameters:
EnType = IncrEn(), embedding dimension = 3, a quantifying resolution = 6, normalization = true.

```
X = ExampleData('randintegers');

Mobj = MSobject('IncrEn', 'm', 3, 'R', 6, 'Norm', true)
>>> Mobj = struct with fields:
    Func: @IncrEn
       m: 3
       R: 6
    Norm: 1
```

Calculate the multiscale increment entropy over 5 temporal scales using the **modified** graining procedure where,

$$y_j^{(\tau)} = \frac{1}{\tau} \sum_{i=(j-1)\tau+1}^{j\tau} x_i \ , \quad 1 \leq j \leq \frac{N}{\tau}$$

```
MSx = MSEn(X, Mobj, 'Scales', 5, 'Methodx', 'modified')

 . . . . . .
>>> MSx = 1×5
    4.2719    4.3059    4.2863    4.2494    4.2773
```

4.1.7 Example 7: Refined Multiscale [Sample] Entropy

Import a signal of uniformly distributed pseudorandom integers in the range [1, 8] and create a multiscale entropy object with the following parameters:
EnType = SampEn(), embedding dimension = 4, radius threshold = 1.25

```
X = ExampleData('randintegers');

Mobj = MSobject('SampEn', 'm', 4, 'r', 1.25)

>>> Mobj = struct with fields:
    Func: @SampEn
       m: 4
       r: 1.2500
```

Calculate the refined multiscale sample entropy and the complexity index (Ci) over 5 temporal scales using a 3rd order Butterworth filter with a normalised corner frequency of at each temporal scale (τ), where the radius threshold value (r) specified by Mobj becomes scaled by the median absolute deviation of the filtered signal at each scale.

```
[MSx, Ci] = rMSEn(X, Mobj, 'Scales', 5, 'F_Order', 3, ...
   'F_Num', 0.6, 'RadNew', 4)

 . . . . . .
>>>MSx = 1×5
    0.5280    0.5734    0.5939    0.5908    0.5563
   Ci = 2.8424
```

4.1.8 Example 8: Composite Multiscale Cross-[Approximate] Entropy

Import two signals of uniformly distributed pseudorandom integers in the range [1 8] and create a multiscale entropy object with the following parameters:
EnType = XApEn(), embedding dimension = 2, time delay = 2, radius distance threshold = 0.5.

```
X = ExampleData('randintegers2');

Mobj = MSobject('XApEn', 'm', 2, 'tau', 2, 'r', 0.5)

>>> Mobj = struct with fields:
    Func: @XApEn
       m: 2
     tau: 2
       r: 0.5000
```

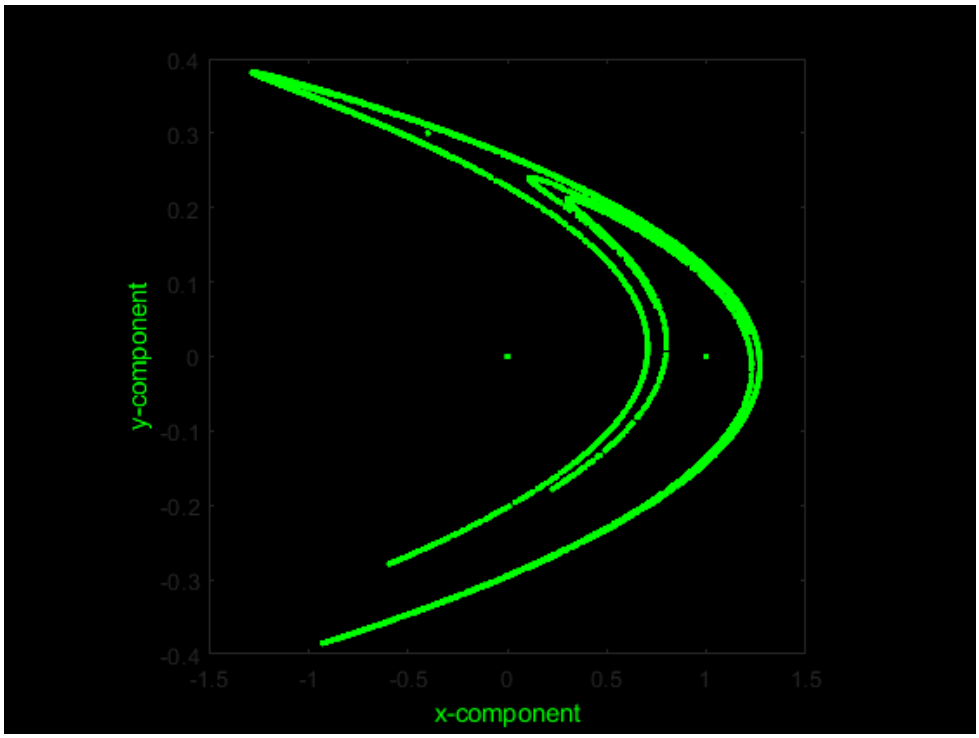
Calculate the comsposite multiscale cross-approximate entropy over 3 temporal scales where the radius distance threshold value (r) specified by Mobj becomes scaled by the variance of the signal at each scale.

```
MSx = cXMSEn(X(:,1), X(:,2), Mobj, 'Scales', 3, 'RadNew', 1)

 . . . . . .
>>> MSx = 1×3
    1.0893    1.4746    1.2932
```

4.1.9 Example 9: Hierarchical Multiscale *corrected* Cross-[Conditional] Entropy

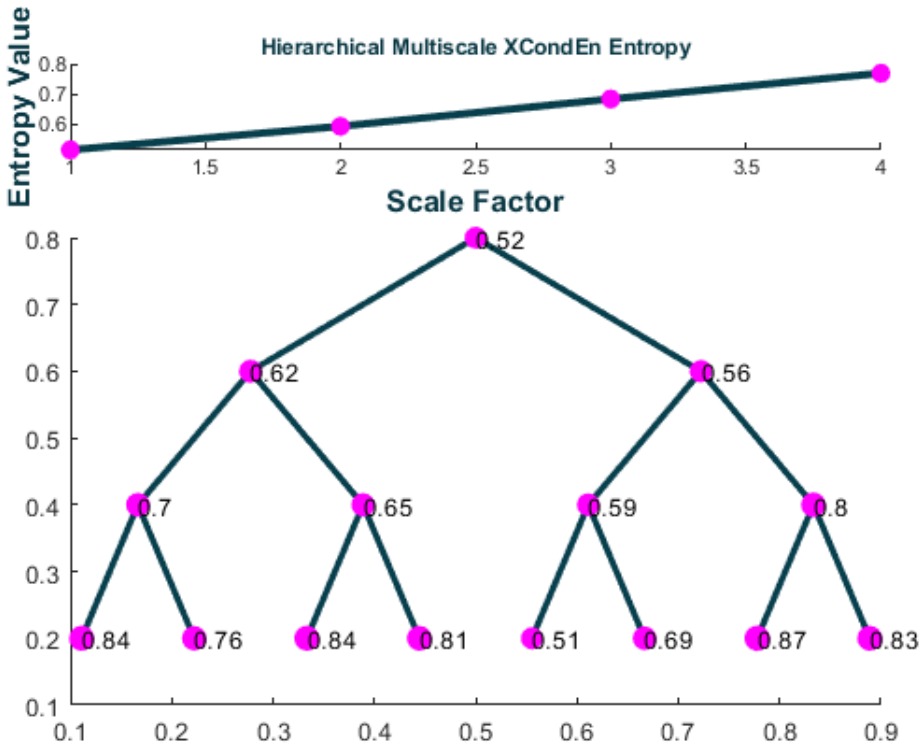
Import the x and y components of the Henon system of equations and create a multiscale entropy object with the following parameters:
EnType = XCondEn(), embedding dimension = 2, time delay = 2, number of symbols = 12, logarithm base = 2, normalization = true

```
Data = ExampleData('henon');

figure('Color', 'k')
plot(Data(:,1), Data(:,2), 'g.')
xlabel('x-component','color','g')
ylabel('y-component','color','g')
set(gca,'color','k'), axis square

Mobj = MSobject('XCondEn', 'm', 2, 'tau', 2, ...
  'c', 12, 'Logx', 2, 'Norm', true)
```



Calculate the hierarchical multiscale corrected cross-conditional entropy over 4 temporal scales and return the average cross-entropy at each scale (Sn), the complexity index (Ci), and a plot of the multiscale entropy curve and the hierarchical tree with the cross-entropy value at each node.

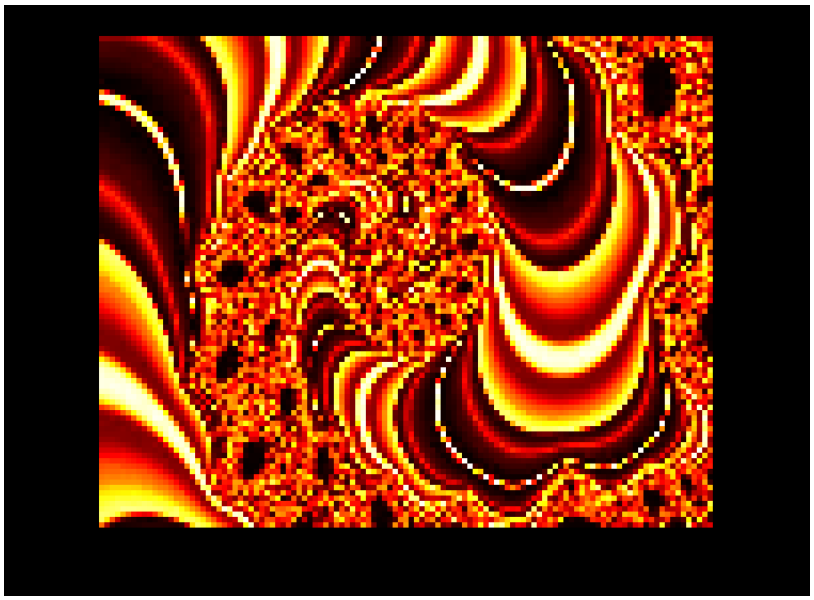
```
[MSx, Sn, Ci] = hXMSEn(Data(:,1), Data(:,2), Mobj, 'Scales', ...
         4, 'Plotx', true)

>>> Only first 4096 samples were used in
   hierarchical decomposition.
>>> The last 404 samples of each data sequence were ignored.

>>> MSx = 1×15
    0.5159    0.6245    0.5634    0.7022    0.6533
      0.5853    0.7956    0.8447    0.7605    0.8415
        0.8115    0.5128    0.6862    0.8679    0.8287
Sn = 1×4
    0.5159    0.5940    0.6841    0.7692
Ci =
2.5632
```



4.1.10 Example 10: Bidimensional Fuzzy Entropy

Import an image of a Mandelbrot fractal as a matrix.

```
X = ExampleData('mandelbrot_Mat');

figure('Color','k'),
imshow(X,[],'InitialMagnification',500),
colormap('hot')
```



Calculate the bidimensional fuzzy entropy in trits (logarithm base 3) with a template matrix of size [8 x 5], and a time delay (tau) of 2 using a 'gaussian' fuzzy function shaped by the parameter **`r = 20`** .

```
FE2D = FuzzEn2D(X, 'm', [8 5], 'tau', 2, 'Fx', . . .
        'gaussian', 'r', 20, 'Logx', 3)

>>> FE2D =
        1.8659
```

4.1.11 Example 11: Multivariate Dispersion Entropy

Import a vector of 4096 uniformly distributed random integers in range [1 8] and convert it to a multivariate set of 4 sequences with 1024 samples each.

```
X = ExampleData('randintegers');
Data = reshape(X, 1024, 4);
```

Calculate the multivariate dispersion entropy and reverse dispersion entropy for embedding dimensions (m) = [1,1,2,3], using a 7-symbol transform.

```
[MDisp, RDE] = MvDispEn(Data, 'm', [1,1,2,3], 'c', 7)

>>> MDisp =
    6.9227345

>>> RDE =
    0.0009856
```

Perform the same calculation but normalize the output entropy estimate w.r.t the number of unique dispersion patterns.

```
[MDisp, RDE] = MvDispEn(Data, 'm', [1,1,2,3], 'c', 7, ...
                        'Norm', true)
>>> MDisp =
    0.508226
>>> RDE =
    0.0009856
```

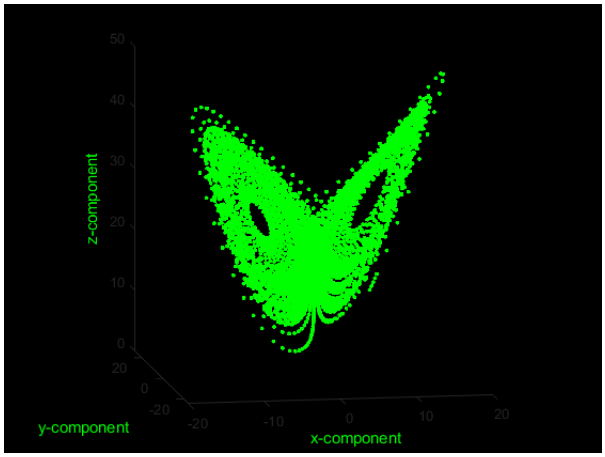
Compare the results above (**Methodx=='v1'**) with those obtained using the ***mvDE*** method (**Methodx=='v2'**), returning estimates for each value from **1,...,max(m)**

```
[MDisp, RDE] = MvDispEn(Data, 'm', [1,1,2,3], 'c', 7, ...
            'Norm', true, 'Methodx', 'v2')
>>> MDisp =
    0.95439595, 0.94074854, 0.93012334
>>> RDE =
    0.02675949, 0.00805324, 0.00201614
```

4.1.12 Example 12: [Generalized] Refined-composite Multivariate Multiscale Fuzzy Entropy

Import the x, y, and z components of the Lorenz system of equations.

```
Data = ExampleData('lorenz');
figure('Color', 'k')

plot3(Data(:,1), Data(:,2), Data(:,3), 'g.')
xlabel('x-component','color','g'),
ylabel('y-component','color','g'),
zlabel('z-component','color','g'),
view(-10,10), set(gca,'color','k'), axis square
```



Create a multiscale entropy object with the following parameters: **`EnType = MvFuzzEn()`**, fuzzy membership function = **`'constgaussian'`**, fuzzy function parameter = 1.75, normalized data to unit variance = true

```
Mobj = MSobject('MvFuzzEn', 'Fx', 'constgaussian',...
                'r', 1.75, 'Norm', true)
```

ATTENTION

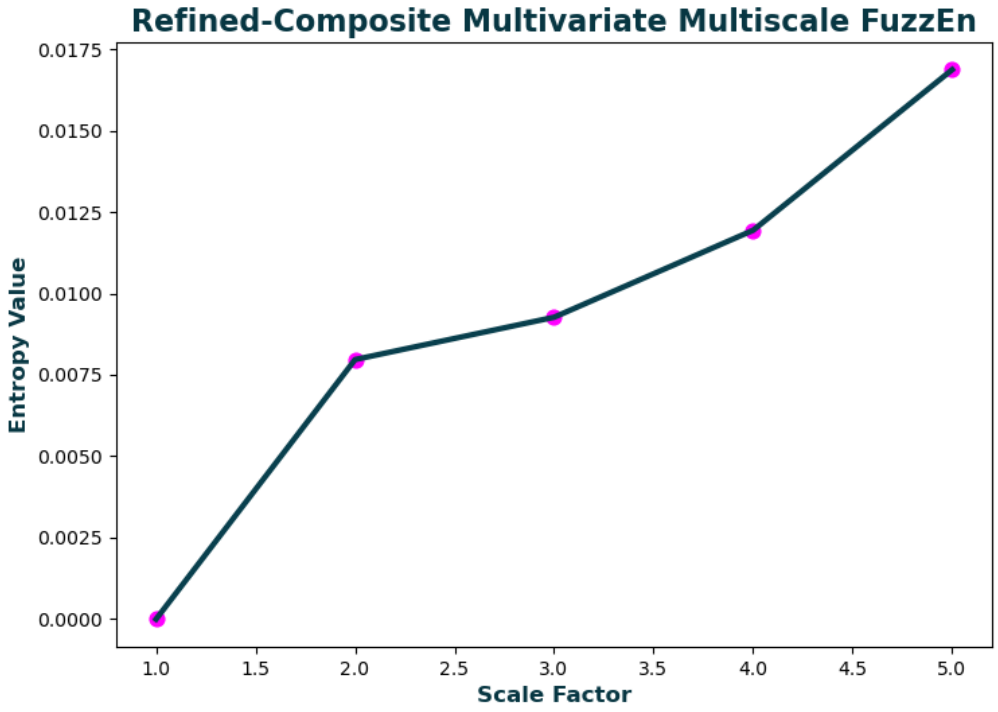
When the multivariate entropy method is multivariate fuzzy entropy (**`MvFuzzEn`**), **`cMvMSEn`** by default employs a generalized graining procedure with the standard deviation (not the variance like in **`MvMSEn`**). This follows the method presented by Azami et. al.[a]

[a] Azami, Fernández and Escudero,
Refined multiscale fuzzy entropy based on standard deviation for biomedical signal analysis
Medical & biological engineering & computing 55 (2017): 2037-2052.

NOTE

As with conventional generalized multiscale entropy, the multiscale entropy value for the first scale will always == 0, as the variance or standard deviation of a singular value is 0!

```
[MSx, CI] = cMvMSEn(Data, Mobj, 'Scales', 5, ...
                            'Refined', true, 'Plotx', true)
>>> MSx =
    0,  0.00796833,  0.00926765,  0.01193731,  0.01686631
>>> RDE =
    0.04603960
```



4.1.13 Example 13: Windowing Data with WindowData()

Create a sequence of integers from 1 - 1000 and segment the values into windows of length 75, with no overlap.

```
X = 1:1000
[WinData, Log] = WindowData(X, WinLen = 75)

>>> WinData =
  {[ 1,  2,  3,  4,  5, . . . 71, 72, 73, 74, 75];
   [ 76,  77,  78,  79, 80, . . .  146, 147, 148, 149, 150];
   [ 151, 152, 153, 154, 155, . . . 221, 222, 223, 224, 225];
   [ 226, 227, 228, 229, 230, . . . 296, 297, 298, 299, 300];
   [ 301, 302, 303, 304, 305, . . . 371, 372, 373, 374, 375];
   [ 376, 377, 378, 379, 380, . . . 446, 447, 448, 449, 450];
   [ 451, 452, 453, 454, 455, . . . 521, 522, 523, 524, 525];
   [ 526, 527, 528, 529, 530, . . . 596, 597, 598, 599, 600];
   [ 601, 602, 603, 604, 605, . . . 671, 672, 673, 674, 675];
   [ 676, 677, 678, 679, 680, . . . 746, 747, 748, 749, 750];
   [ 751, 752, 753, 754, 755, . . . 821, 822, 823, 824, 825];
   [ 826, 827, 828, 829, 830, . . . 896, 897, 898, 899, 900];
   [ 901, 902, 903, 904, 905, . . . 971, 972, 973, 974, 975]}

>>> Log =
     struct with fields:
     DataType: "single univariate vector (1 sequence)"
   DataLength: 1000
 WindowLength: 75
WindowOverlap: 0
 TotalWindows: 13
         Mode: "exlcude"
```

Repeat the previous step, but include any remaining values that do not fill the final window.

```
X = 1:1000
[WinData, Log] = WindowData(X, WinLen = 75)

>>> WinData =
  {[ 1,  2,  3,  4,  5, . . . 71, 72, 73, 74, 75];
   [ 76,  77,  78,  79, 80, . . .  146, 147, 148, 149, 150];
   [ 151, 152, 153, 154, 155, . . . 221, 222, 223, 224, 225];
   [ 226, 227, 228, 229, 230, . . . 296, 297, 298, 299, 300];
   [ 301, 302, 303, 304, 305, . . . 371, 372, 373, 374, 375];
   [ 376, 377, 378, 379, 380, . . . 446, 447, 448, 449, 450];
   [ 451, 452, 453, 454, 455, . . . 521, 522, 523, 524, 525];
   [ 526, 527, 528, 529, 530, . . . 596, 597, 598, 599, 600];
   [ 601, 602, 603, 604, 605, . . . 671, 672, 673, 674, 675];
   [ 676, 677, 678, 679, 680, . . . 746, 747, 748, 749, 750];
   [ 751, 752, 753, 754, 755, . . . 821, 822, 823, 824, 825];
   [ 826, 827, 828, 829, 830, . . . 896, 897, 898, 899, 900];
   [ 901, 902, 903, 904, 905, . . . 971, 972, 973, 974, 975]}
   [ 976, 977, 978, 979, 980, . . . 996, 997, 998, 999, 1000]}
  %% ^ Remaining values included in final array (N = 25) ^

>>> Log =
      struct with fields:
     DataType: "single univariate vector (1 sequence)"
   DataLength: 1000
 WindowLength: 75
WindowOverlap: 0
 TotalWindows: 14 %% < Total windows increased from 13 to 14
         Mode: "include"
```

Create a matrix of 4 sequences of integers (1:1000, 1001:2000, 2001:3000, 3001:4000). Segment the data into windows of length 130x4, with 20 samples overlap.

```
X = [1:1000; 1001:2000; 2001:3000; 3001:4000]' ;
[WinData, Log] = WindowData(X, WinLen = 130, Overlap = 20)

>>> WinData =
    {[1,   1001,  2001,  3001
      2,   1002,  2002,  3002
      3,   1003,  2003,  3003
             ...
      128,  1128,  2128,  3128
      129,  1129,  2129,  3129
      130,  1130,  2130,  3130];
     [111,  1111,  2111,  3111
      112,  1112,  2112,  3112
      113,  1113,  2113,  3113
             ...
      238,  1238,  2238,  3238
      239,  1239,  2239,  3239
      240,  1240,  2240,  3240];
        :     :       :     :
        :     :       :     :
        :     :       :     :
        :     :       :     :
     [771,  1771,  2771,  3771
      772,  1772,  2772,  3772
      773,  1773,  2773,  3773
             ...
      898,  1898,  2898,  3898
      899,  1899,  2899,  3899
      900,  1900,  2900,  3900]}

>>> Log =
     struct with fields:
     DataType: "multivariate matrix (4 vectors)"
   DataLength: 1000
 WindowLength: 130
WindowOverlap: 20
 TotalWindows: 8
         Mode: "exlcude"
```

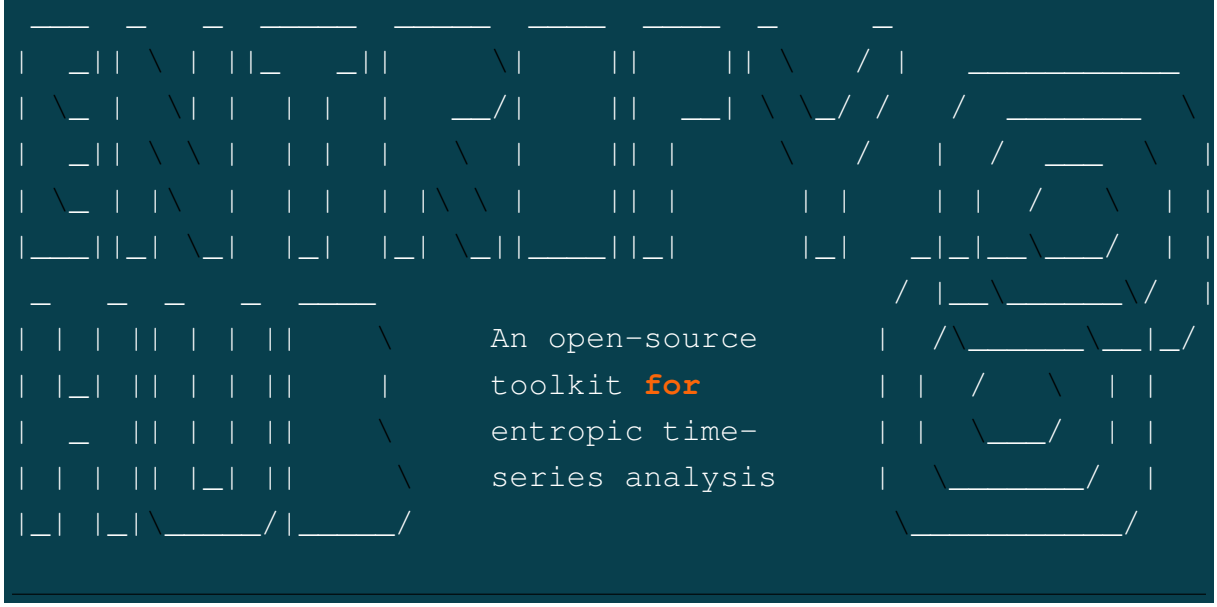
4.2 Python:

After EntropyHub has been installed in python, it must be imported in order to use it.

```
import EntropyHub
```

In the following python examples, it is assumed that EntropyHub has been imported using the 'eh' abbreviation:

```
import EntropyHub as eh
eh.greet()
```



NOTE

Python functions in EntropyHub are based primarily on the `Numpy` module. Arguments in python functions with the **`np.`** prefix refer to `numpy` functions.

4.2.1 Example 1: Sample Entropy

Import a signal of normally distributed random numbers [$\mu = 0, \sigma = 1$], and calculate the sample entropy for each embedding dimension (m) from 0 to 4.

```
X = eh.ExampleData('gaussian');

Samp, Phi1, Phi2 = eh.SampEn(X, m = 4)

>>> Samp =
    array([2.1789, 2.1757, 2.1819, 2.2209, 2.1756])
```

Select the last value to get the sample entropy for m = 4.

```
Samp[-1]

>>> 2.1756
```

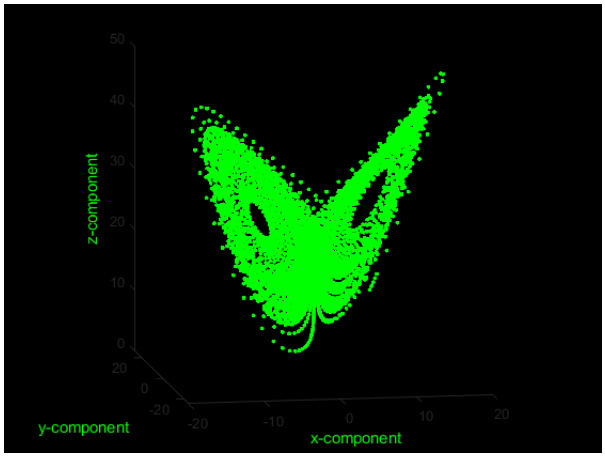
Calculate the sample entropy for each embedding dimension (m) from 0 to 4 with a time delay (tau) of 2 samples.

```
Samp, Phi1, Phi2 = eh.SampEn(X, m = 4, tau = 2)

>>> Samp =
    array([2.1789    2.1833    2.1880    2.1892    2.1441])
```

4.2.2 Example 2: (Fine-Grained) Permutation Entropy

Import the x, y, and z components of the Lorenz system of equations.

```
Data = eh.ExampleData('lorenz');
from matplotlib.pyplot import fig, scatter, axis
fig = figure(facecolor='k')
ax = fig.add_subplot(111, projection='3d')
ax.set_facecolor('k')
ax.scatter(Data[:,0], Data[:,1], Data[:,2], c='g')
ax.axis('off')
```



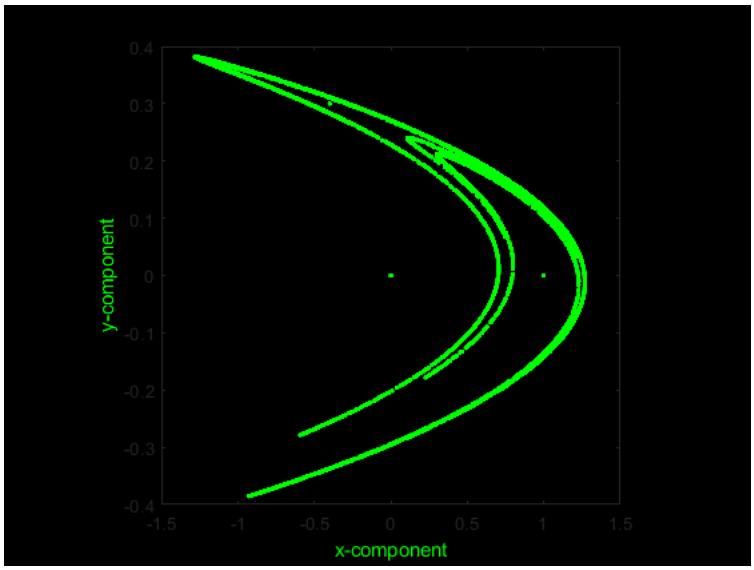
Calculate fine-grained permutation entropy of the z component in dits (logarithm base 10) with an embedding dimension of 3, time delay of 2, an alpha parameter of 1.234. Return `Pnorm` normalised w.r.t the number of all possible permutations (m!) and the condition permutation entropy (`cPE`) estimate.

```
Z = Data[:,2];

Perm, Pnorm, cPE = eh.PermEn(Z, m = 3, tau = 2,
Typex = 'finegrain', tpx = 1.234, Logx = 10, Norm = False)

>>> Perm
        array([-0.   ,  0.8687,  0.9468])
>>> Pnorm
        array([ nan, 0.8687, 0.4734])
>>> cPE
        array([0.8687, 0.0781])
```

4.2.3 Example 3: Phase Entropy w/ Poincaré plot

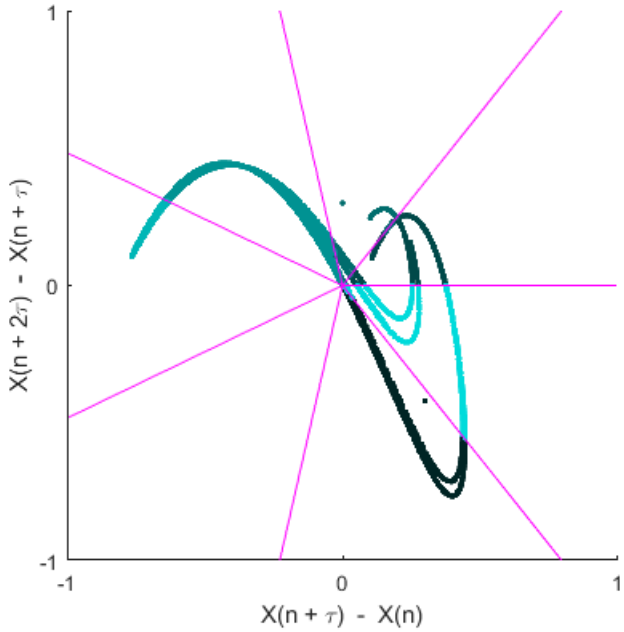
Import the x and y components of the Henon system of equations.

```
from matplotlib.pyplot import figure, plot, axis

Data = eh.ExampleData('henon');
fig = figure(facecolor='k')
plot(Data[:,0], Data[:,1], 'g.')
axis('off')
```



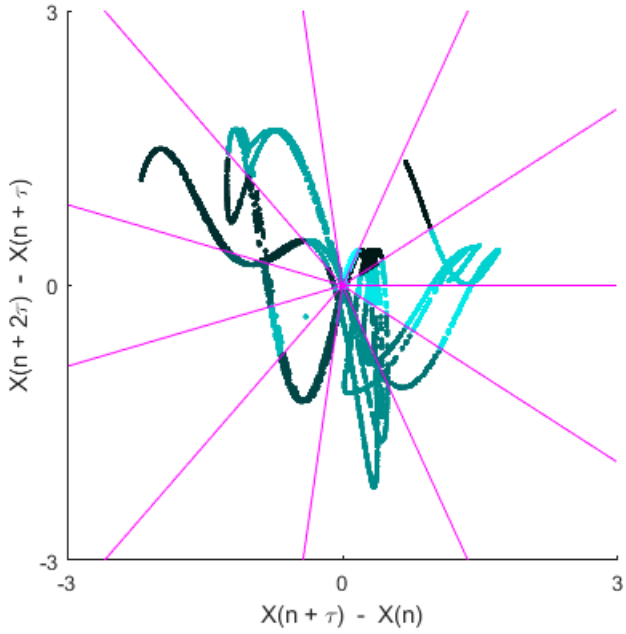
Calculate the phase entropy of the y-component in bits (logarithm base 2) without normalization using 7 angular partitions and return the second-order difference plot.

```
Y = Data[:,1];
Phas = eh.PhasEn(Y, K = 7, Norm = False, Logx = 2,
                 Plotx = True)

>>> Phas
        2.0192821496913216
```



Calculate the phase entropy of the x-component using 11 angular partitions, a time delay of 2, and return the second-order difference plot.

```
X = Data[:,0]
Phas = eh.PhasEn(X, K = 11, tau = 2, Plotx = True)

>>> Phas
        0.8395
```



4.2.4 Example 4: Cross-Distribution Entropy w/ Different Binning Methods

Import a signal of pseudorandom integers in the range [1, 8] and calculate the cross-distribution entropy with an embedding dimension of 5, a time delay (tau) of 3, and Sturges' bin selection method.

```
X = eh.ExampleData('randintegers2');

XDist, _ = eh.XDistEn(X[:,0], X[:,1], m = 5, tau = 3)

>>> Note: 17/25 bins were empty
  XDist =
    0.5248
```

Use Rice's method to determine the number of histogram bins and return the probability of each bin (Ppi).

```
XDist, Ppi = eh.XDistEn(X[:,0], X[:,1], m = 5, tau = 3,
         Bins = 'rice')

>>> Note: 407/415 bins were empty
>>> XDist =
      0.2802
>>> Ppi =
      array([0.0000    0.0047    0.0368    0.1096
                0.1978    0.2558    0.2421    0.1531])
```

4.2.5 Example 5: Multiscale Entropy Object [MSobject()]

Create a multiscale entropy object (Mobj) for multiscale fuzzy entropy, calculated with an embedding dimension of 5, a time delay of 2, using a sigmoidal fuzzy function with the r scaling parameters (3, 1.2).

```
Mobj = eh.MSobject('FuzzEn', m = 5, tau = 2,
                         Fx = 'sigmoid', r = (3, 1.2))

Mobj.Func
>>> <function EntropyHub._FuzzEn.FuzzEn(Sig, m=2,
        tau=1, r=(0.2, 2), Fx='default', Logx=2.71828)>

Mobj.Kwargs
>>> {'m': 5, 'tau': 2, 'Fx': 'sigmoid', 'r': (3, 1.2)}
```

Create a multiscale entropy object (Mobj) for multiscale corrected-cross-conditional entropy, calculated with an embedding dimension of 6 and using a 11-symbolic data transform.

```
Mobj = eh.MSobject('XCondEn', m = 6, c = 11)

Mobj.Func
>>> <function EntropyHub._XCondEn.XCondEn(Sig, m=2,
         tau=1, c=6, Logx=2.71828, Norm=False)>

Mobj.Kwargs
>>> {'m': 6, 'c': 11}
```

4.2.6 Example 6: Multiscale [Increment] Entropy

Import a signal of uniformly distributed pseudorandom integers in the range [1,8] and create a multiscale entropy object with the following parameters:
EnType = IncrEn(), embedding dimension = 3, a quantifying resolution = 6, normalization = true.

```
X = eh.ExampleData('randintegers');

Mobj = eh.MSobject('IncrEn', m = 3, R = 6, Norm = True)

Mobj.Func
>>> <function EntropyHub._IncrEn.IncrEn(Sig,
                  m=2, tau=1, R=4, Logx=2, Norm=False)>

Mobj.Kwargs
>>> {'m': 3, 'R': 6, 'Norm': True}
```

Calculate the multiscale increment entropy over 5 temporal scales using the **modified** graining procedure where,

$$y_j^{(\tau)} = \frac{1}{\tau} \sum_{i=(j-1)\tau+1}^{j\tau} x_i \ , \quad 1 \leq j \leq \frac{N}{\tau}$$

```
MSx, _ = eh.MSEn(X, Mobj, Scales = 5, Methodx = 'modified')

 . . . . . .
>>> MSx =
    array([4.2719   4.3059   4.2863   4.2494   4.2773])
```

4.2.7 Example 7: Refined Multiscale [Sample] Entropy

Import a signal of uniformly distributed pseudorandom integers in the range [1, 8] and create a multiscale entropy object with the following parameters:
EnType = SampEn(), embedding dimension = 4, radius threshold = 1.25

```
X = eh.ExampleData('randintegers');

Mobj = eh.MSobject('SampEn', m = 4, r = 1.25)

Mobj.Func
>>> <function EntropyHub._SampEn.SampEn(Sig, m=2,
                        tau=1, r=None, Logx=2.71828)>

Mobj.Kwargs
>>> {'m': 4, 'r': 1.25}
```

Calculate the refined multiscale sample entropy and the complexity index (Ci) over 5 temporal scales using a 3rd order Butterworth filter with a normalised corner frequency of at each temporal scale (τ), where the radius threshold value (r) specified by Mobj becomes scaled by the median absolute deviation of the filtered signal at each scale.

```
MSx, Ci = eh.rMSEn(X, Mobj, Scales = 5, F_Order = 3,
                F_Num = 0.6, RadNew = 4)

 . . . . . .
>>> MSx
        array([0.5280, 0.5734, 0.5940, 0.5908, 0.5564])

>>> Ci
        2.842518179468045
```

4.2.8 Example 8: Composite Multiscale Cross-[Approximate] Entropy

Import two signals of uniformly distributed pseudorandom integers in the range [1 8] and create a multiscale entropy object with the following parameters:
EnType = XApEn(), embedding dimension = 2, time delay = 2, radius distance threshold = 0.5.

```
X = eh.ExampleData('randintegers2');

Mobj = eh.MSobject('XApEn', m = 2, tau = 2, r = 0.5)

Mobj.Func
>>> <function EntropyHub._XApEn.XApEn(Sig, m=2,
                          tau=1, r=None, Logx=2.71828)>

Mobj.Kwargs
>>> {'m': 2, 'tau': 2, 'r': 0.5}
```

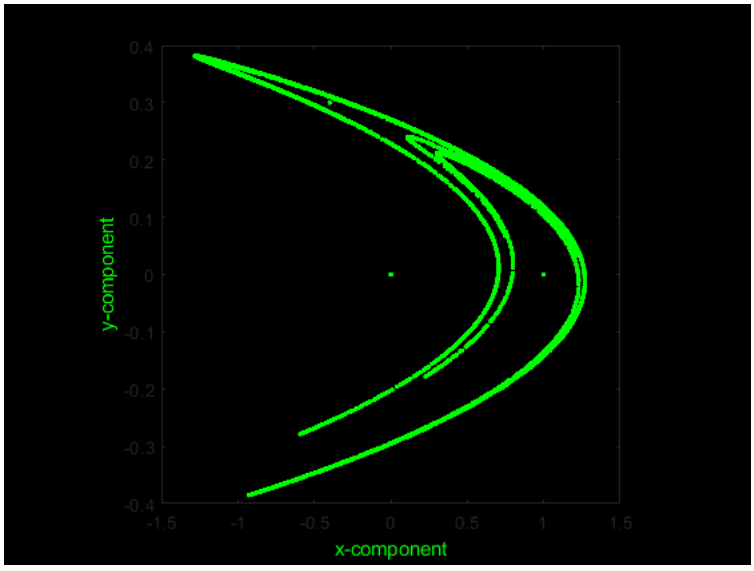
Calculate the comsposite multiscale cross-approximate entropy over 3 temporal scales where the radius distance threshold value (r) specified by Mobj becomes scaled by the variance of the signal at each scale.

```
MSx, _ = eh.cXMSEn(X[:,0], X[:,1], Mobj, Scales = 3,
                         RadNew = 1)

 . . . . . .
>>> MSx =
  array([1.089, 1.4746, 1.2932])
```

4.2.9 Example 9: Hierarchical Multiscale *corrected* Cross-[Conditional] Entropy

Import the x and y components of the Henon system of equations and create a multiscale entropy object with the following parameters:
EnType = XCondEn(), embedding dimension = 2, time delay = 2, number of symbols = 12, logarithm base = 2, normalization = true

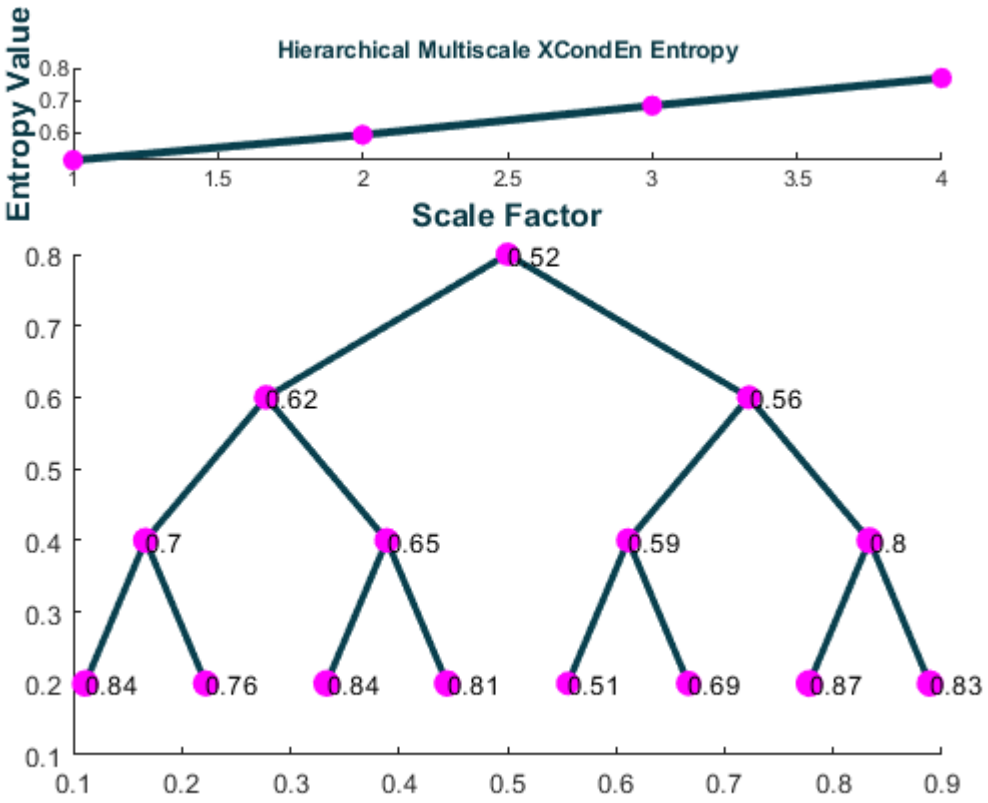
```
from matplotlib.pyplot import figure, plot, axis

Data = eh.ExampleData('henon');

fig = figure(facecolor='k')
plot(Data[:,0], Data[:,1], 'g.')
axis('off')

Mobj = eh.MSobject('XCondEn', m = 2, tau = 2, c = 12,
                   Logx = 2, Norm = True)

Mobj.Func
>>> <function EntropyHub._XCondEn.XCondEn(Sig, m=2,
                   tau=1, c=6, Logx=2.71828, Norm=False)>

Mobj.Kwargs
>>> {'m': 2, 'tau': 2, 'c': 12, 'Logx': 2, 'Norm': True}
```



Calculate the hierarchical multiscale corrected cross-conditional entropy over 4 temporal

scales and return the average cross-entropy at each scale (Sn), the complexity index (Ci), and a plot of the multiscale entropy curve and the hierarchical tree with the cross-entropy value at each node.

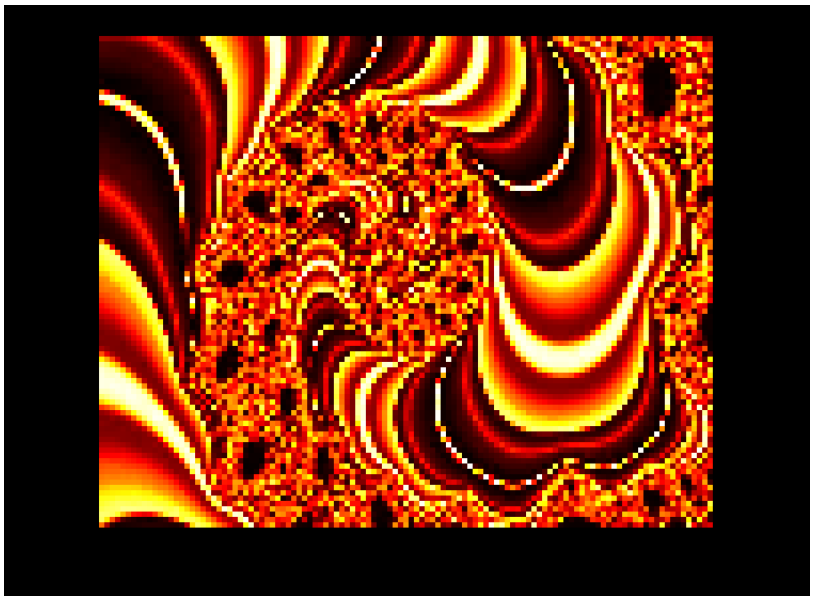
```
MSx, Sn, Ci = eh.hXMSEn(Data[:,0], Data[:,1], Mobj,
                Scales = 4, Plotx = True)

>>> WARNING: Only first 4096 samples were used in
   hierarchical decomposition.
The last 404 samples of each data sequence were ignored.
. . . . . .
>>> MSx =
    array([0.5159, 0.6245, 0.5634, 0.7022, 0.6533,
                    0.5853, 0.7956, 0.8447, 0.7605, 0.8415,
                     0.8115, 0.5128, 0.6862, 0.8679, 0.8287])
Sn =
    array([0.5159, 0.5940, 0.6841, 0.7692])
Ci =
        2.5632
```



4.2.10 Example 10: Bidimensional Fuzzy Entropy

Import an image of a Mandelbrot fractal as a matrix.

```
X = eh.ExampleData('mandelbrot_Mat');

from matplotlib.pyplot import imshow, show

imshow(X, cmap = 'hot'), show()
```



Calculate the bidimensional fuzzy entropy in trits (logarithm base 3) with a template matrix of size [8 x 5], and a time delay (tau) of 2 using a 'gaussian' fuzzy function shaped by the parameter `r = 20`.

```
FE2D = eh.FuzzEn2D(X, m = (8, 5), tau = 2, Fx = 'gaussian',
                          r = 20, Logx = 3)

>>> FE2D =
        1.8659
```

4.2.11 Example 11: Multivariate Dispersion Entropy

Import a vector of 4096 uniformly distributed random integers in range [1 8] and convert it to a multivariate set of 4 sequences with 1024 samples each.

```
X = eh.ExampleData('randintegers')
Data = np.reshape(X,(4,1024)).T
```

Calculate the multivariate dispersion entropy and reverse dispersion entropy for embedding dimensions (m) = [1,1,2,3], using a 7-symbol transform.

```
MDisp, RDE = eh.MvDispEn(Data, m = np.array([1,1,2,3]), c = 7)
>>> MDisp =
    6.9227345
>>> RDE =
    0.0009856
```

Perform the same calculation but normalize the output entropy estimate w.r.t the number of unique dispersion patterns.

```
MDisp, RDE = eh.MvDispEn(Data, m = np.array([1,1,2,3]), c = 7,
                         Norm = True)
>>> MDisp =
    0.508226
>>> RDE =
    0.0009856
```

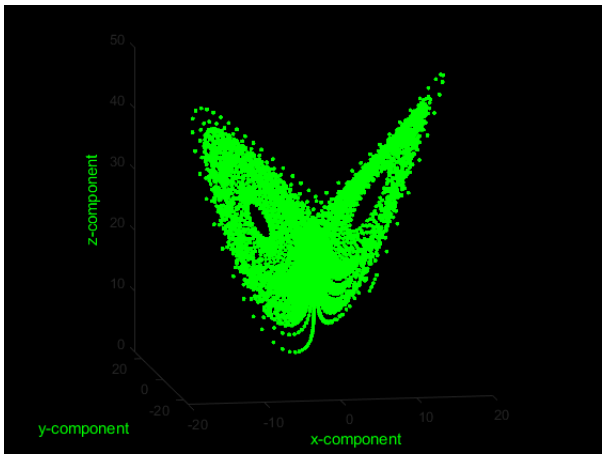
Compare the results above (**Methodx=='v1'**) with those obtained using the ***mvDE*** method (**Methodx=='v2'**), returning estimates for each value from **1,...,max(m)**

```
MDisp, RDE = eh.MvDispEn(Data, m = np.array([1,1,2,3]), c = 7,
                         Norm = True, Methodx = 'v2')

>>> MDisp =
    0.95439595, 0.94074854, 0.93012334
>>> RDE =
    0.02675949, 0.00805324, 0.00201614
```

4.2.12 Example 12: [Generalized] Refined-composite Multivariate Multiscale Fuzzy Entropy

Import the x, y, and z components of the Lorenz system of equations.

```
Data = eh.ExampleData('lorenz');
from matplotlib.pyplot import fig, scatter, axis
fig = figure(facecolor='k')
ax = fig.add_subplot(111, projection='3d')
ax.set_facecolor('k')
ax.scatter(Data[:,0], Data[:,1], Data[:,2], c='g')
ax.axis('off')
```



Create a multiscale entropy object with the following parameters: **EnType = MvFuzzEn()**, fuzzy membership function = **'constgaussian'**, fuzzy function parameter = 1.75, normalized data to unit variance = true

```
Mobj = eh.MSobject('MvFuzzEn', Fx = 'constgaussian',
                   r = 1.75, Norm = True)
```

ATTENTION

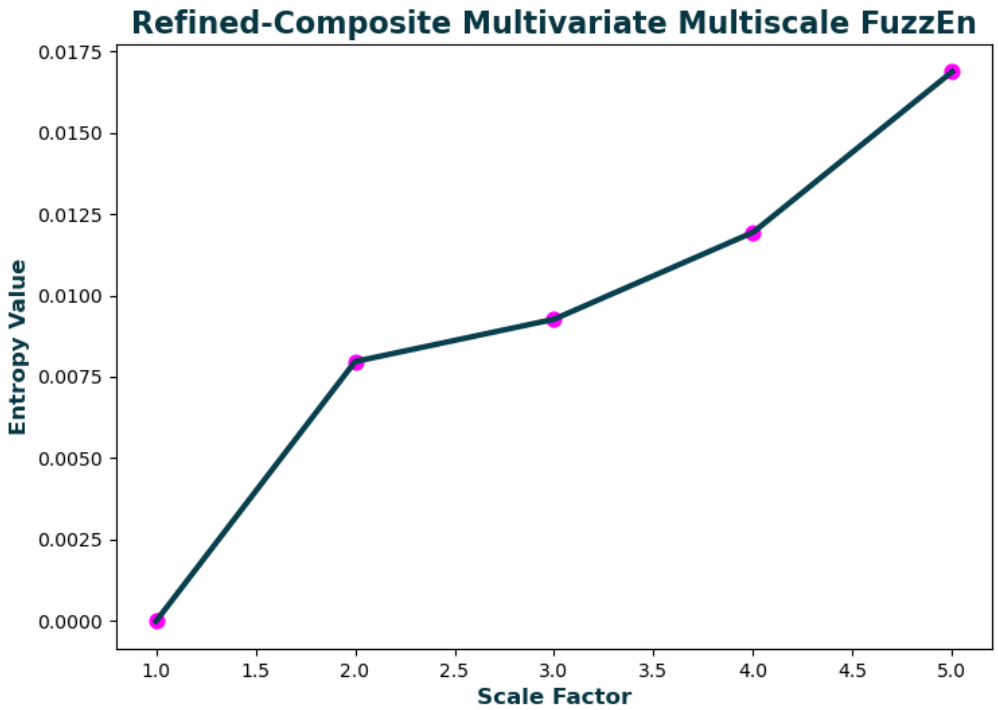
When the multivariate entropy method is multivariate fuzzy entropy (**MvFuzzEn**), **cMvMSEn** by default employs a generalized graining procedure with the standard deviation (not the variance like in **MvMSEn**). This follows the method presented by Azami et. al.[a]

[a] Azami, Fernández and Escudero,
Refined multiscale fuzzy entropy based on standard deviation for biomedical signal analysis
Medical & biological engineering & computing 55 (2017): 2037-2052.

NOTE

As with conventional generalized multiscale entropy, the multiscale entropy value for the first scale will always == 0, as the variance or standard deviation of a singular value is 0!

```
MSx, CI = eh.cMvMSEn(Data, Mobj, Scales = 5,
                Refined = True, Plotx = True)

>>> MSx =
    0,  0.00796833,  0.00926765,  0.01193731,  0.01686631

>>> RDE =
    0.04603960
```



4.2.13 Example 13: Windowing Data with WindowData()

Create a sequence of integers from 1 - 1000 and segment the values into windows of length 75, with no overlap.

```
X = np.arange(1,1001)
WinData, Log = eh.WindowData(X, WinLen = 75)

>>> WinData =
(array([ 1,  2,  3,  4,  5, ... 71, 72, 73, 74, 75]),
 array([ 76,  77,  78,  79, 80, ...  146, 147, 148, 149, 150]),
 array([151, 152, 153, 154, 155, ... 221, 222, 223, 224, 225]),
 array([226, 227, 228, 229, 230, ... 296, 297, 298, 299, 300]),
 array([301, 302, 303, 304, 305, ... 371, 372, 373, 374, 375]),
 array([376, 377, 378, 379, 380, ... 446, 447, 448, 449, 450]),
 array([451, 452, 453, 454, 455, ... 521, 522, 523, 524, 525]),
 array([526, 527, 528, 529, 530, ... 596, 597, 598, 599, 600]),
 array([601, 602, 603, 604, 605, ... 671, 672, 673, 674, 675]),
 array([676, 677, 678, 679, 680, ... 746, 747, 748, 749, 750]),
 array([751, 752, 753, 754, 755, ... 821, 822, 823, 824, 825]),
 array([826, 827, 828, 829, 830, ... 896, 897, 898, 899, 900]),
 array([901, 902, 903, 904, 905, ... 971, 972, 973, 974, 975]))

>>> Log =
    {'DataType': 'single univariate vector (1 sequence)',
     'DataLength': 1000,
     'WindowLength': 75,
     'WindowOverlap': 0,
     'TotalWindows': 13,
     'Mode': 'exclude'}
```

Repeat the previous step, but include any remaining values that do not fill the final window.

```
WinData, Log = eh.WindowData(X, WinLen = 75, Mode = 'include')

>>> WinData =
(array([ 1,  2,  3,  4,  5, ... 71, 72, 73, 74, 75]),
 array([ 76,  77,  78,  79, 80, ...  146, 147, 148, 149, 150]),
 array([151, 152, 153, 154, 155, ... 221, 222, 223, 224, 225]),
 array([226, 227, 228, 229, 230, ... 296, 297, 298, 299, 300]),
 array([301, 302, 303, 304, 305, ... 371, 372, 373, 374, 375]),
 array([376, 377, 378, 379, 380, ... 446, 447, 448, 449, 450]),
 array([451, 452, 453, 454, 455, ... 521, 522, 523, 524, 525]),
 array([526, 527, 528, 529, 530, ... 596, 597, 598, 599, 600]),
 array([601, 602, 603, 604, 605, ... 671, 672, 673, 674, 675]),
 array([676, 677, 678, 679, 680, ... 746, 747, 748, 749, 750]),
 array([751, 752, 753, 754, 755, ... 821, 822, 823, 824, 825]),
 array([826, 827, 828, 829, 830, ... 896, 897, 898, 899, 900]),
 array([901, 902, 903, 904, 905, ... 971, 972, 973, 974, 975]),
 array([976, 977, 978, 979, 980, ... 996, 997, 998, 999, 1000])
    # ^ Remaining values included in final array (N = 25) ^

>>> Log =
  {'DataType': 'single univariate vector (1 sequence)',
   'DataLength': 1000,
   'WindowLength': 75,
   'WindowOverlap': 0,
   'TotalWindows': 14, #<- Total windows increased from 13 to 14
   'Mode': 'include'}
```

Create a matrix of 4 sequences of integers (1:1000, 1001:2000, 2001:3000, 3001:4000). Segment the data into windows of length 130x4, with 20 samples overlap.

```
X = np.vstack((np.arange(1,1001), np.arange(1001,2001),
               np.arange(2001,3001), np.arange(3001,4001))).T

WinData, Log = eh.WindowData(X, WinLen = 130, Overlap = 20)

>>> WinData =
    (array([1,  1001,  2001,  3001
            2,  1002,  2002,  3002
            3,  1003,  2003,  3003
            ...
            128,  1128,  2128,  3128
            129,  1129,  2129,  3129
            130,  1130,  2130,  3130]),
     array([111,  1111,  2111,  3111
            112,  1112,  2112,  3112
            113,  1113,  2113,  3113
            ...
            238,  1238,  2238,  3238
            239,  1239,  2239,  3239
            240,  1240,  2240,  3240]),
             :     :       :     :
             :     :       :     :
             :     :       :     :
     array([771,  1771,  2771,  3771
            772,  1772,  2772,  3772
            773,  1773,  2773,  3773
            ...
            898,  1898,  2898,  3898
            899,  1899,  2899,  3899
            900,  1900,  2900,  3900]))

>>> Log =
    {'DataType': 'multivariate matrix (4 vectors)',
     'DataLength': 1000,
     'WindowLength': 130,
     'WindowOverlap': 20,
     'TotalWindows': 8,
     'Mode': 'exclude'}
```

4.3 Julia:

After EntropyHub has been installed in Julia, it must be imported in order to use it.

```
using EntropyHub
```

In the following julia examples, it is assumed that EntropyHub has already been imported. To check that EntropyHub is active in your julia REPL, type:

```
julia> EntropyHub.greet()


 ____  _   _  _____  _____  ____  ____  _     _
|  _|| \ | ||_   _||     \|    ||    || \   / |   ____________
| \_ |  \| |  | |  |   __/|    ||  __| \ \_/ /   /  ________  \
|  _|| \ \ |  | |  |    \  |    || |     \   /    |  /  ___  \  |
| \_ | |\  |  | |  | |\ \ |    || |      | |     | |  /   \  | |
|___||_| \_|  |_|  |_| \_||____||_|      |_|     _|_|__\___/  | |
 _   _  _   _  ____                              / |__\_______\/  |
| | | || | | ||    \      An open-source       |  /\_______\__|_/
| |_| || | | ||     |     toolkit for          | |  /   \  | |
|  _  || | | ||    \      entropic time-       | |  \___/  | |
| | | || |_| ||     \     series analysis       |  \_______/  |
|_| |_|\_____/|_____/                            \____________/
```

NOTE

Some functions have the option to return a plot of the results, e.g. `PhasEn()`, `GridEn()`, `MSEn()`, etc.
Make sure that you are using the correct plotting backend for your IDE before returning plots through EntropyHub functions.

4.3.1 Example 1: Sample Entropy

Import a signal of normally distributed random numbers $[\mu = 0, \sigma = 1]$, and calculate the sample entropy for each embedding dimension (m) from 0 to 4.

```
julia> X = ExampleData("gaussian");

julia> Samp, _ = SampEn(X, m = 4)
([2.17892361, 2.17574232, 2.1819695, 2.22098397, 2.175566717])
```

Select the last value to get the sample entropy for m = 4.

```
julia> Samp[end]
 2.178923612371957
```

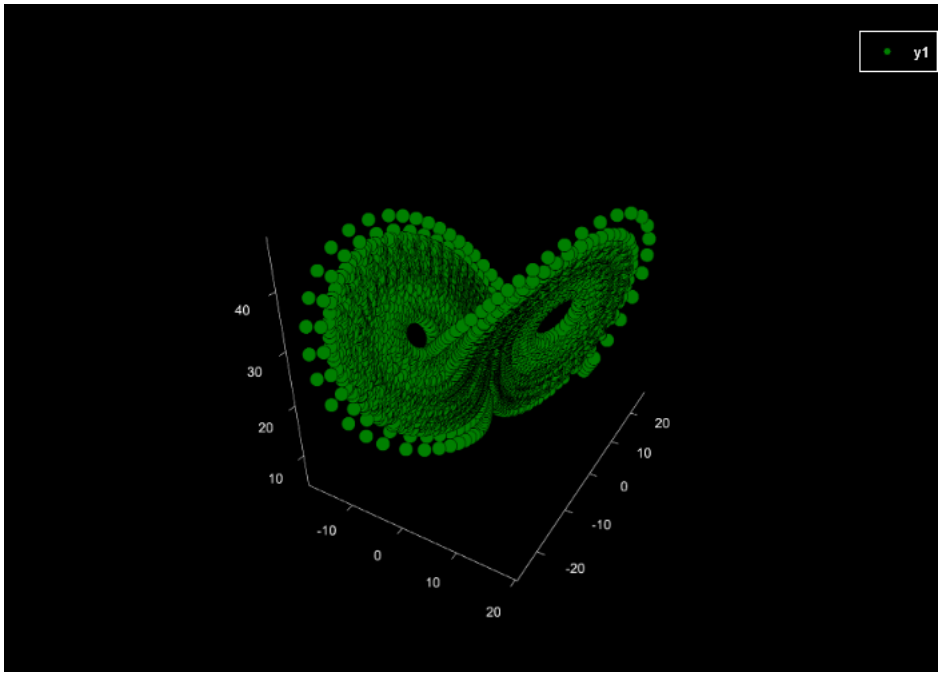
Calculate the sample entropy for each embedding dimension (m) from 0 to 4 with a time delay (tau) of 2 samples.

```
julia> Samp, Phi1, Phi2 = SampEn(X, m = 4, tau = 2)
([2.17892361, 2.18332325, 2.18804107, 2.189184333, 2.1440802],
[1.414258e6, 159224.0, 17843.0, 1998.0, 234.0],
[1.24975e7, 1.413233e6, 159119.0, 17838.0, 1997.0])
```

4.3.2 Example 2: (Fine-Grained) Permutation Entropy

Import the x, y, and z components of the Lorenz system of equations.

```
julia> Data = ExampleData("lorenz");

julia> using Plots
julia> Plots.backend()   # Check that the right backend is in use!

julia> scatter(Data[:,1], Data[:,2], Data[:,3],
markercolor = "green", markerstrokecolor = "black",
markersize = 3, background_color = "black", grid = false)
```



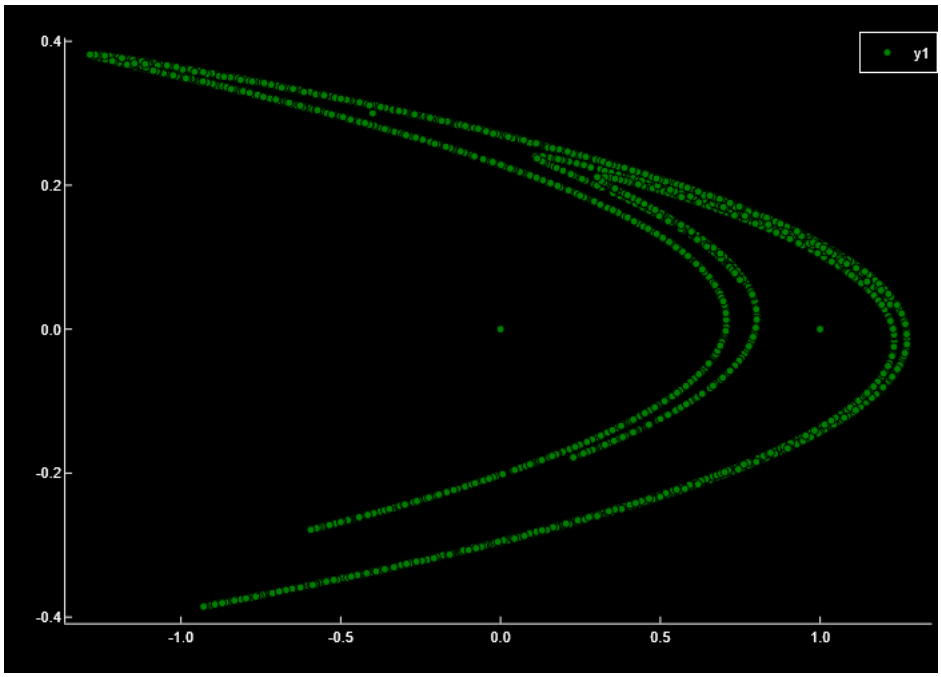
Calculate fine-grained permutation entropy of the z component in dits (logarithm base 10) with an embedding dimension of 3, time delay of 2, an alpha parameter of 1.234. Return `Pnorm` normalised w.r.t the number of all possible permutations (m!) and the condition permutation entropy (`cPE`) estimate.

```
julia> Z = Data[:,3];
julia> Perm, Pnorm, cPE = PermEn(Z, m = 3, tau = 2,
Typex = "finegrain", tpx = 1.234, Logx = 10, Norm = false)

([-0.0, 0.8686539340402203, 0.946782979031713],
[NaN, 0.8686539340402203, 0.4733914895158565],
[0.8686539340402203, 0.07812904499149276])
```

4.3.3 Example 3: Phase Entropy w/ Poincaré plot

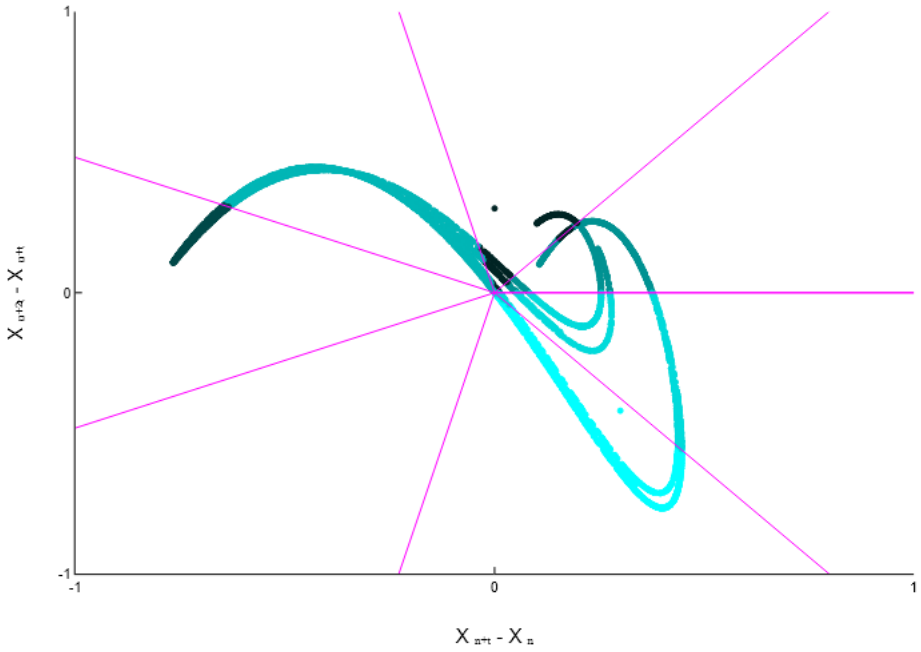
Import the x and y components of the Henon system of equations.

```
julia> using Plots
julia> Plots.backend()   # Check that the right backend is in use!
julia> Data = ExampleData("henon");

julia> scatter(Data[:,1], Data[:,2],
markercolor = "green", markerstrokecolor = "black",
markersize = 3, background_color = "black",grid = false)
```



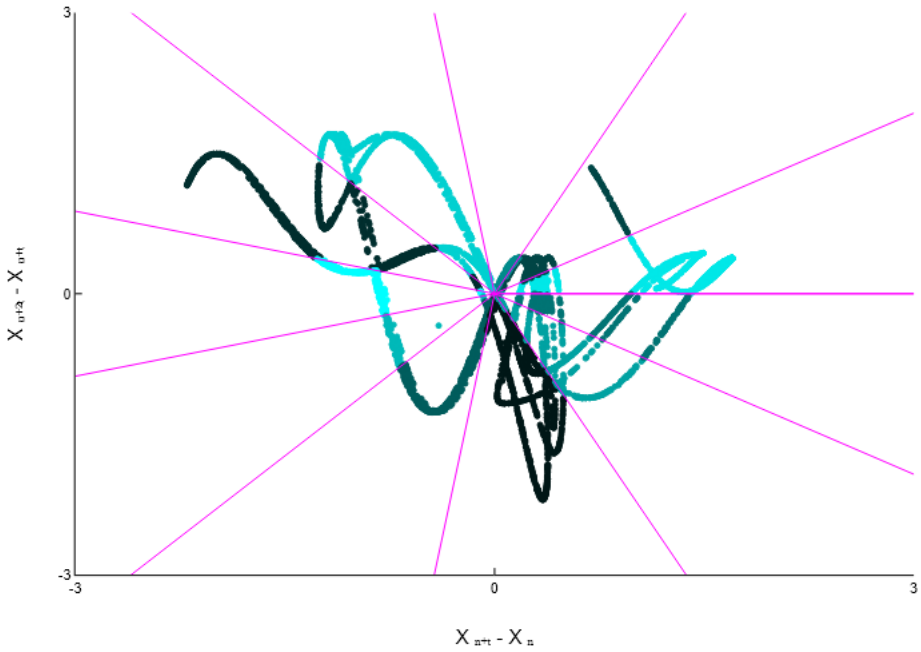
Calculate the phase entropy of the y-component in bits (logarithm base 2) without normalization using 7 angular partitions and return the second-order difference plot.

```
julia> Y = Data[:,2];
julia> Phas = PhasEn(Y, K = 7, Norm = false, Logx = 2,
        Plotx = true)
2.0193
```



Calculate the phase entropy of the x-component using 11 angular partitions, a time delay of 2, and return the second-order difference plot.

```
julia> X = Data[:,1];
julia> Phas = PhasEn(X, K = 11, tau = 2, Plotx = true)
0.8395391613164361
```



4.3.4 Example 4: Cross-Distribution Entropy w/ Different Binning Methods

Import a signal of pseudorandom integers in the range [1, 8] and calculate the cross-distribution entropy with an embedding dimension of 5, a time delay (tau) of 3, and Sturges' bin selection method.

```
julia> X = ExampleData("randintegers2");
julia> XDist, _ = XDistEn(X[:,1], X[:,2], m = 5, tau = 3)
Note: 17/25 bins were empty
0.524841365239631
```

Use Rice's method to determine the number of histogram bins and return the probability of each bin (Ppi).

```
julia> XDist, Ppi = XDistEn(X[:,1], X[:,2], m = 5, tau = 3,
                 Bins = "rice")
Note: 407/415 bins were empty
(0.28024570808915084,
[3.59537211e-5, 0.004693585, 0.03679902, 0.1095869, 0.1978132,
    0.25581946, 0.24212389, 0.1531279])
```

4.3.5 Example 5: Multiscale Entropy Object [MSobject()]

Note: Unlike MatLab or Python, in Julia the base and cross- entropy functions used in the multiscale entropy calculation are declared by passing EntropyHub **functions** to `MSobject()`, not string names.

Create a multiscale entropy object (Mobj) for multiscale fuzzy entropy, calculated with an embedding dimension of 5, a time delay of 2, using a sigmoidal fuzzy function with the r scaling parameters (3, 1.2).

```
julia> Mobj = MSobject(FuzzEn, m = 5, tau = 2,
                           Fx = "sigmoid", r = (3, 1.2))
(Func = EntropyHub._FuzzEn.FuzzEn, m = 5, tau = 2,
                    Fx = "sigmoid", r = (3, 1.2))
```

Create a multiscale entropy object (Mobj) for multiscale corrected-cross-conditional entropy, calculated with an embedding dimension of 6 and using a 11-symbolic data transform.

```
julia> Mobj = MSobject(XCondEn, m = 6, c = 11)
(Func = EntropyHub._XCondEn.XCondEn, m = 6, c = 11)
```

4.3.6 Example 6: Multiscale [Increment] Entropy

Import a signal of uniformly distributed pseudorandom integers in the range [1,8] and create a multiscale entropy object with the following parameters:
EnType = IncrEn(), embedding dimension = 3, a quantifying resolution = 6, normalization = true.

```
julia> X = ExampleData("randintegers");
julia> Mobj = MSobject(IncrEn, m = 3, R = 6, Norm = true);
julia> Mobj
(Func = EntropyHub._IncrEn.IncrEn, m = 3, R = 6, Norm = true)
```

Calculate the multiscale increment entropy over 5 temporal scales using the **modified** graining procedure where,

$$y_j^{(\tau)} = \frac{1}{\tau} \sum_{i=(j-1)\tau+1}^{j\tau} x_i \,, \quad 1 \leq j \leq \frac{N}{\tau}$$

```
julia> MSx, _ = MSEn(X, Mobj, Scales = 5, Methodx = "modified")
 . . . . . .
([4.2719   4.3059   4.2863   4.2494   4.2773])
```

4.3.7 Example 7: Refined Multiscale [Sample] Entropy

Import a signal of uniformly distributed pseudorandom integers in the range [1, 8] and create a multiscale entropy object with the following parameters:
EnType = SampEn(), embedding dimension = 4, radius threshold = 1.25

```
julia> X = ExampleData("randintegers");
julia> Mobj = MSobject(SampEn, m = 4, r = 1.25)
(Func = EntropyHub._SampEn.SampEn, m = 4, r = 1.25)
```

Calculate the refined multiscale sample entropy and the complexity index (Ci) over 5 temporal scales using a 3rd order Butterworth filter with a normalised corner frequency of at each temporal scale (τ), where the radius threshold value (r) specified by Mobj becomes scaled by the median absolute deviation of the filtered signal at each scale.

```
julia> MSx, Ci = rMSEn(X, Mobj, Scales = 5, F_Order = 3,
                F_Num = 0.6, RadNew = 4)
. . . . .
([0.52796539, 0.57338645, 0.593936, 0.5907829, 0.5564473],
2.842518179)
```

4.3.8 Example 8: Composite Multiscale Cross-[Approximate] Entropy

Import two signals of uniformly distributed pseudorandom integers in the range [1 8] and create a multiscale entropy object with the following parameters:
EnType = XApEn(), embedding dimension = 2, time delay = 2, radius distance threshold = 0.5.

```
julia> X = ExampleData("randintegers2");
julia> Mobj = MSobject(XApEn, m = 2, tau = 2, r = 0.5)
(Func = EntropyHub._XApEn.XApEn, m = 2, tau = 2, r = 0.5)
```

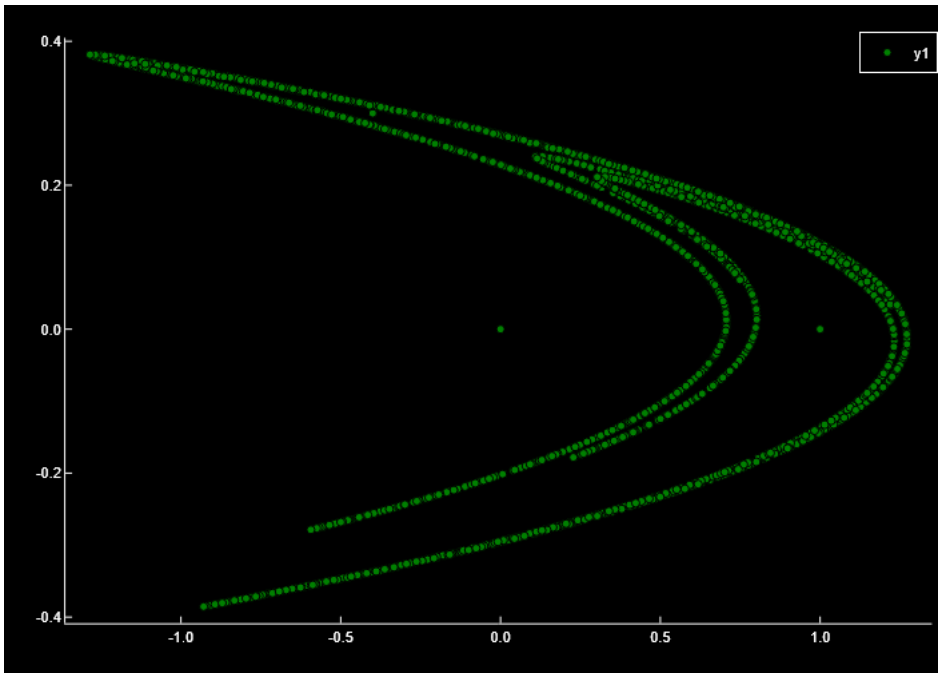
Calculate the comsposite multiscale cross-approximate entropy over 3 temporal scales where the radius distance threshold value (r) specified by Mobj becomes scaled by the variance of the signal at each scale.

```
julia> MSx, _ = cXMSEn(X[:,1], X[:,2], Mobj, Scales = 3,
                RadNew = 1)

 . . . . . .
[1.0893229452569062, 1.4745638145624824, 1.293182408488266]
```

4.3.9 Example 9: Hierarchical Multiscale *corrected* Cross-[Conditional] Entropy

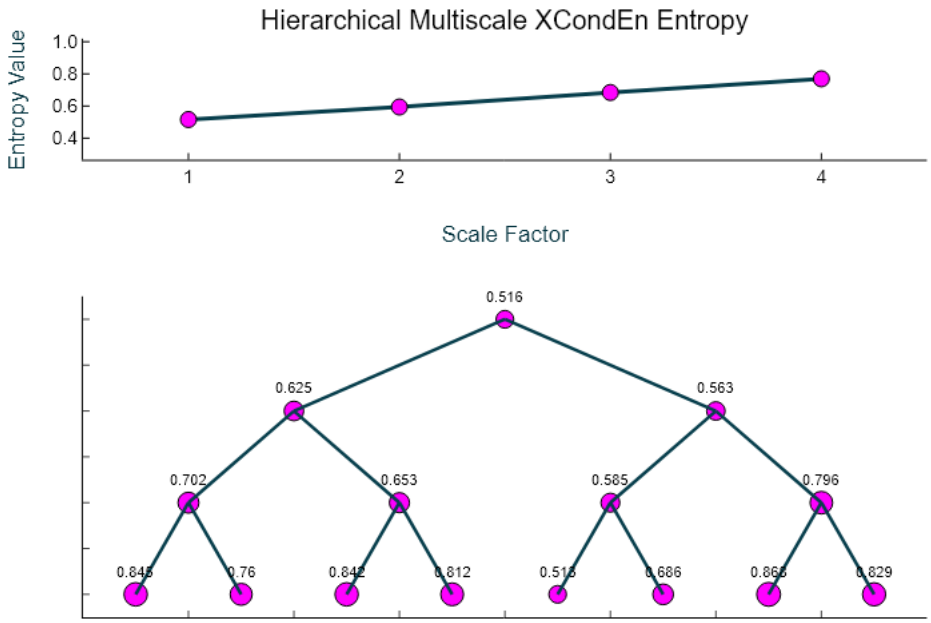
Import the x and y components of the Henon system of equations and create a multiscale entropy object with the following parameters:
EnType = XCondEn(), embedding dimension = 2, time delay = 2, number of symbols = 12, logarithm base = 2, normalization = true

```
julia> using Plots
julia> Plots.backend()  # Check that the right backend is in use!
julia> Data = ExampleData("henon");

julia> scatter(Data[:,1], Data[:,2],
markercolor = "green", markerstrokecolor = "black",
markersize = 3, background_color = "black",grid = false)

julia> Mobj = MSobject(XCondEn, m = 2, tau = 2, c = 12,
                Logx = 2, Norm = true)
(Func = EntropyHub._XCondEn.XCondEn, m = 2, tau = 2, c = 12,
         Logx = 2, Norm = true)
```



Calculate the hierarchical multiscale corrected cross-conditional entropy over 4 temporal scales and return the average cross-entropy at each scale (Sn), the complexity index (Ci), and a plot of the multiscale entropy curve and the hierarchical tree with the cross-entropy value at each node.

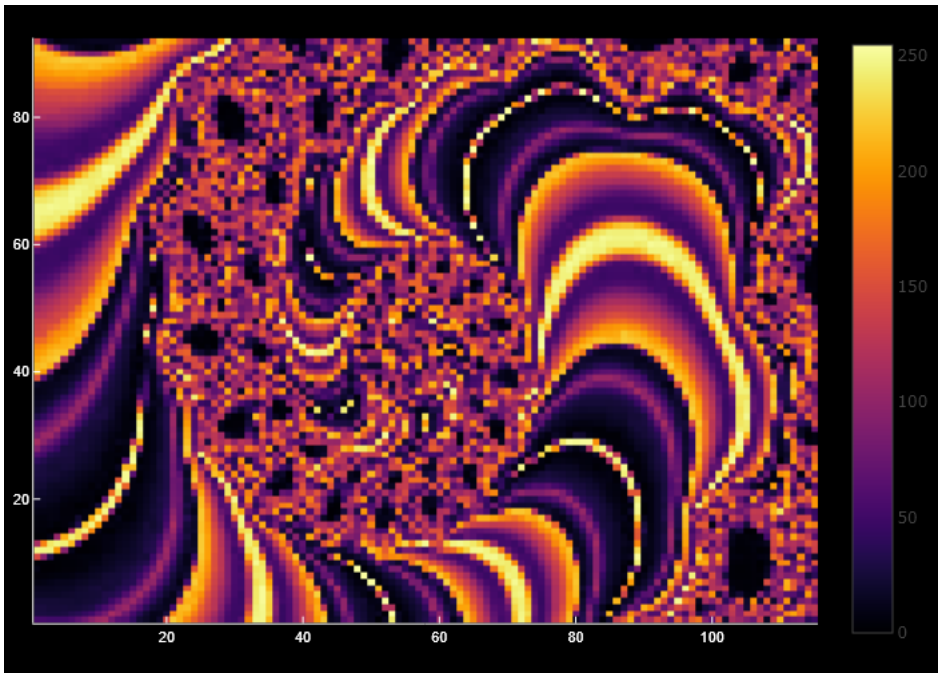
```
MSx, Sn, Ci = hXMSEn(Data[:,1], Data[:,2], Mobj, Scales = 4,
                Plotx = true)
[Warning: Only first 4096 samples were used in hierarchical
|         decomposition.
|    The last 404 samples of the data sequence were ignored.
[@ EntropyHub._hXMSEn
. . . . . .
([0.5159119, 0.62451155, 0.563417, 0.7022124, 0.653264,
0.58528238, 0.7956453, 0.8446734, 0.7604554, 0.8415218,
0.81153266, 0.51284941, 0.68619314, 0.86785005, 0.8287299],
[0.5159119, 0.59396428, 0.68410104, 0.76922575],
2.5632030151318776)
```



4.3.10 Example 10: Bidimensional Fuzzy Entropy

Import an image of a Mandelbrot fractal as a matrix.

```
julia> using Plots
julia> Plots.backend()   # Check that the right backend is in use!

julia> X = ExampleData("mandelbrot_Mat");
julia> heatmap(X, background_color="black")
```



Calculate the bidimensional fuzzy entropy in trits (logarithm base 3) with a template matrix of size [8 x 5], and a time delay (tau) of 2 using a 'gaussian' fuzzy function shaped by the parameter `r = 20`.

```
julia> FE2D = FuzzEn2D(X, m = (8, 5), tau = 2,
                       Fx = "gaussian", r = 20, Logx = 3)
1.8659
```

4.3.11 Example 11: Multivariate Dispersion Entropy

Import a vector of 4096 uniformly distributed random integers in range [1 8] and convert it to a multivariate set of 4 sequences with 1024 samples each.

```
julia>  X = ExampleData("randintegers")
julia>  Data = reshape(X, 1024, 4)
```

Calculate the multivariate dispersion entropy and reverse dispersion entropy for embedding dimensions (m) = [1,1,2,3], using a 7-symbol transform.

```
julia>  MDisp, RDE = MvDispEn(Data, m = [1,1,2,3], c = 7)
(6.922734508722953, 0.0009856206548534644)
```

Perform the same calculation but normalize the output entropy estimate w.r.t the number of unique dispersion patterns.

```
julia>  MDisp, RDE = MvDispEn(Data, m = [1,1,2,3], c = 7,
                         Norm = true)
(0.5082259698140852, 0.0009856218516602513)
```

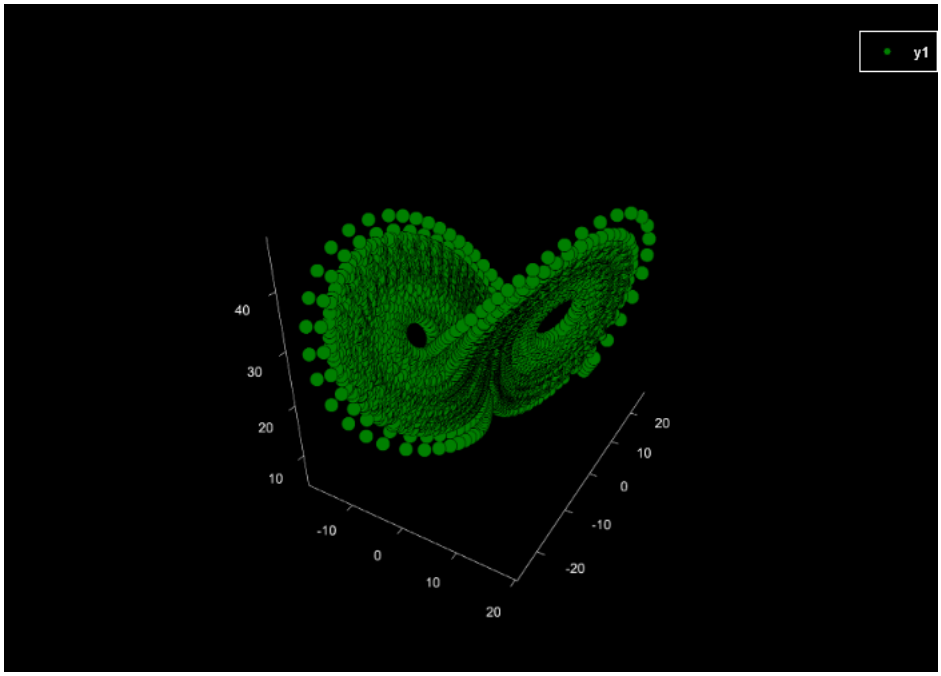
Compare the results above (**Methodx=='v1'**) with those obtained using the ***mvDE*** method (**Methodx=='v2'**), returning estimates for each value from **1,...,max(m)**

```
julia>  MDisp, RDE = MvDispEn(Data, m = [1,1,2,3], c = 7,
                         Norm = true, Methodx = "v2")
 .   .   .
([0.954395954191354, 0.94074853715469, 0.930123339696689],
 [0.026759488075315, 0.00805323667175, 0.0020161358982543])
```

4.3.12 Example 12: [Generalized] Refined-composite Multivariate Multiscale Fuzzy Entropy

Import the x, y, and z components of the Lorenz system of equations.

```
julia> Data = ExampleData("lorenz");

julia> using Plots
julia> Plots.backend()   # Check that the right backend is in use!

julia> scatter(Data[:,1], Data[:,2], Data[:,3],
markercolor = "green", markerstrokecolor = "black",
markersize = 3, background_color = "black", grid = false)
```



Create a multiscale entropy object with the following parameters: **`EnType = MvFuzzEn()`**, fuzzy membership function = **`'constgaussian'`**, fuzzy function parameter = 1.75, normalized data to unit variance = true

```
julia>  Mobj = MSobject(MvFuzzEn, Fx = "constgaussian",
                r = 1.75, Norm = true)
```

ATTENTION

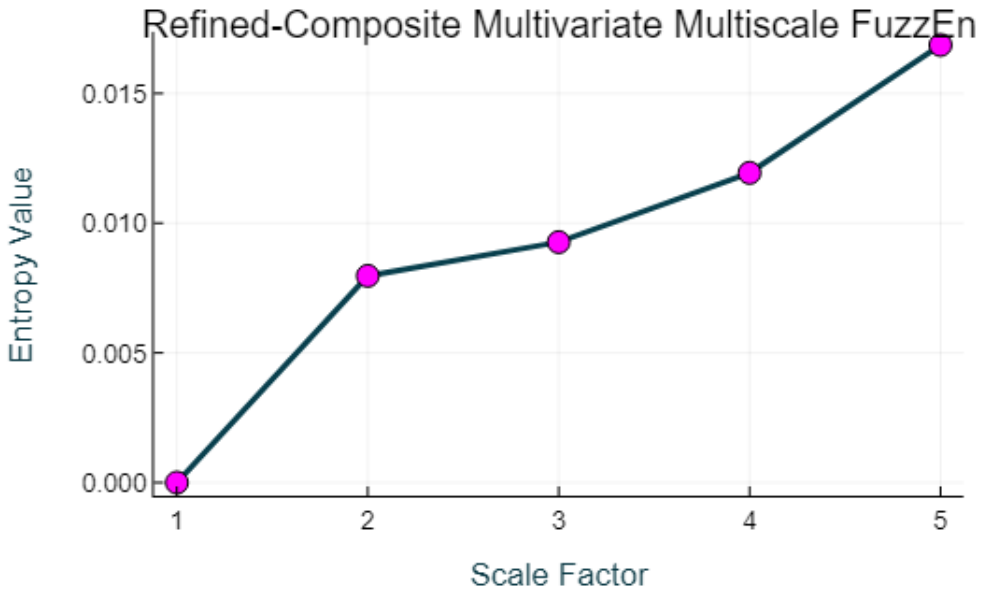
When the multivariate entropy method is multivariate fuzzy entropy (**MvFuzzEn**), **cMvMSEn** by default employs a generalized graining procedure with the standard deviation (not the variance like in **MvMSEn**). This follows the method presented by Azami et. al.[a]

[a] Azami, Fernández and Escudero,
Refined multiscale fuzzy entropy based on standard deviation for biomedical signal analysis
Medical & biological engineering & computing 55 (2017): 2037-2052.

NOTE

As with conventional generalized multiscale entropy, the multiscale entropy value for the first scale will always == 0, as the variance or standard deviation of a singular value is 0!

```
julia>  MSx, CI = cMvMSEn(Data, Mobj, Scales = 5, Refined = tru
                          Plotx = true)
. . . . . . . . . . . . . . . . . . . . . . .
([-0.0, 0.00796833, 0.00926765, 0.01193731, 0.016866306],
 0.046039601)
```



4.3.13 Example 13: Windowing Data with WindowData()

Create a sequence of integers from 1 - 1000 and segment the values into windows of length 75, with no overlap.

```
julia>  X = 1:1001
julia>  WinData, Log = WindowData(X, WinLen = 75)

(Any[[1; 2; ... ; 74; 75;;],
[76; 77; ... ; 149; 150;;],
[151; 152; ... ; 224; 225;;],
[226; 227; ... ; 299; 300;;],
[301; 302; ... ; 374; 375;;],
[376; 377; ... ; 449; 450;;],
[451; 452; ... ; 524; 525;;],
[526; 527; ... ; 599; 600;;],
[601; 602; ... ; 674; 675;;],
[676; 677; ... ; 749; 750;;],
[751; 752; ... ; 824; 825;;],
[826; 827; ... ; 899; 900;;],
[901; 902; ... ; 974; 975;;]],
Dict{String, Any}(
"DataType" => "single univariate vector (1 sequence)",
"TotalWindows" => 13,
"WindowOverlap" => 0,
"Mode" => "exclude",
"WindowLength" => 75,
"DataLength" => 1001))
```

Repeat the previous step, but include any remaining values that do not fill the final window.

```
julia>  WinData, Log = WindowData(X, WinLen = 75,
                        Mode = "include")

(Any[[1; 2; ... ; 74; 75;;],
[76; 77; ... ; 149; 150;;],
[151; 152; ... ; 224; 225;;],
[226; 227; ... ; 299; 300;;],
[301; 302; ... ; 374; 375;;],
[376; 377; ... ; 449; 450;;],
[451; 452; ... ; 524; 525;;],
[526; 527; ... ; 599; 600;;],
[601; 602; ... ; 674; 675;;],
[676; 677; ... ; 749; 750;;],
[751; 752; ... ; 824; 825;;],
[826; 827; ... ; 899; 900;;],
[901; 902; ... ; 974; 975;;],
[976; 977; ... ; 1000; 1001;;]],   # <- Remaining values included
Dict{String, Any}(                 #    in final array (N = 25)
"DataType" => "single univariate vector (1 sequence)",
"TotalWindows" => 14, #<- Total windows increased from 13 to 14
"WindowOverlap" => 0,
"Mode" => "include",
"WindowLength" => 75,
"DataLength" => 1001))
```

Create a matrix of 4 sequences of integers (1:1000, 1001:2000, 2001:3000, 3001:4000). Segment the data into windows of length 130x4, with 20 samples overlap.

```
julia>  X = hcat(1:1001, 1001:2001, 2001:3001, 3001:4001)
julia>  WinData, Log = WindowData(X, WinLen = 130,
                        Overlap = 20)

(Any[[1 1001 2001 3001; ... ; 130 1130 2130 3130],
[111 1111 2111 3111; ... ;  240 1240 2240 3240],
[221 1221 2221 3221; ... ;  350 1350 2350 3350],
[331 1331 2331 3331; ... ;  460 1460 2460 3460],
[441 1441 2441 3441; ... ;  570 1570 2570 3570],
[551 1551 2551 3551; ... ;  680 1680 2680 3680],
[661 1661 2661 3661; ... ;  790 1790 2790 3790],
[771 1771 2771 3771; ... ;  900 1900 2900 3900]],
Dict{String, Any}(
"DataType" => "multivariate matrix (4 vectors)",
"TotalWindows" => 8,
"WindowOverlap" => 20,
"Mode" => "exclude",
"WindowLength" => 130,
"DataLength" => 1001))
```

5

References

[1] Steven M. Pincus,
Approximate entropy as a measure of system complexity,
Proceedings of the National Academy of Sciences, 88.6 (1991): 2297-2301.

[2] Joshua S Richman and J. Randall Moorman,
Physiological time-series analysis using approximate entropy and sample entropy,
American Journal of Physiology-Heart and Circulatory Physiology (2000).

[3] Douglas E Lake, Joshua S Richman, M.P. Griffin, J. Randall Moorman,
Sample entropy analysis of neonatal heart rate variability,
American Journal of Physiology-Regulatory, Integrative and Comparative Physiology, 283, no. 3 (2002): R789-R797.

[4] Weiting Chen, et al.
Characterization of surface EMG signal based on fuzzy entropy,
IEEE Transactions on neural systems and rehabilitation engineering, 15.2 (2007): 266-272.

[5] Hong-Bo Xie, Wei-Xing He, and Hui Liu,
Measuring time series regularity using nonlinear similarity-based sample entropy,
Physics Letters A, 372.48 (2008): 7140-7146.

[6] Hamed Azami, et al.
Fuzzy Entropy Metrics for the Analysis of Biomedical Signals: Assessment and Comparison,
IEEE Access, 7 (2019): 104833-104847

[7] Peter Grassberger and Itamar Procaccia, *Estimation of the Kolmogorov entropy from a chaotic signal*,
Physical review A 28.4 (1983): 2591.

[8] Lin Gao, Jue Wang and Longwei Chen,
Event-related desynchronization and synchronization quantification in motor-related EEG by Kolmogorov entropy,
J Neural Engineering, 10(3) (2013) : 03602

[9] Christoph Bandt and Bernd Pompe,
Permutation entropy: A natural complexity measure for time series,
Physical Review Letters, 88.17 (2002): 174102.

[10] Xiao-Feng Liu, and Wang Yue,
Fine-grained permutation entropy as a measure of natural complexity for time series,
Chinese Physics B, 18.7 (2009): 2690.

[11] Chunhua Bian, et al.,
Modified permutation-entropy analysis of heartbeat dynamics,
Physical Review E, 85.2 (2012) : 021906

[12] Bilal Fadlallah, et al.,
Weighted-permutation entropy: A complexity measure for time series incorporating amplitude information,
Physical Review E, 87.2 (2013): 022911.

[13] Hamed Azami and Javier Escudero,
Amplitude-aware permutation entropy: Illustration in spike detection and signal segmentation,
Computer methods and programs in biomedicine, 128 (2016): 40-51.

[14] Zhiqiang Huo, et al.,
Edge Permutation Entropy: An Improved Entropy Measure for Time-Series Analysis,
45th Annual Conference of the IEEE Industrial Electronics Soc, (2019), 5998-6003

[15] Zhe Chen, et al.,
Improved permutation entropy for measuring complexity of time series under noisy condition,
Complexity, 1403829 (2019).

[16] Maik Riedl, Andreas Müller, and Niels Wessel,
Practical considerations of permutation entropy,
The European Physical Journal Special Topics, 222.2 (2013): 249-262.

[17] Huan Kang, Xiaofeng Zhang, Guangbin Zhang,
Phase permutation entropy: A complexity measure for nonlinear time series incorporating phase information
Physica A: Statistical Mechanics and its Applications 568 (2021): 125686.

[18] Alberto Porta, et al.,
Measuring regularity by means of a corrected conditional entropy in sympathetic outflow,
Biological cybernetics 78.1 (1998): 71-78.

[19] Li, Peng, et al.,
Assessing the complexity of short-term heartbeat interval series by distribution entropy,
Medical & biological engineering & computing 53.1 (2015): 77-87.

[20] G.E. Powell and I.C. Percival,
A spectral entropy method for distinguishing regular and irregular motion of Hamiltonian systems.
Journal of Physics A: Mathematical and General 12.11 (1979): 2053.

[21] Tsuyoshi Inouye, et al.,
Quantification of EEG irregularity by use of the entropy of the power spectrum,
Electroencephalography and clinical neurophysiology 79.3 (1991): 204-210.

[22] Mostafa Rostaghi and Hamed Azami,
Dispersion entropy: A measure for time-series analysis
IEEE Signal Processing Letters 23.5 (2016): 610-614.

[23] Hamed Azami and Javier Escudero,
Amplitude-and fluctuation-based dispersion entropy,
Entropy 20.3 (2018): 210.

[24] Li Yuxing, Xiang Gao and Long Wang,
Reverse dispersion entropy: A new complexity measure for sensor signal,
Sensors 19.23 (2019): 5203.

[25] Wenlong Fu, et al.,
Fault diagnosis for rolling bearings based on fine-sorted dispersion entropy and SVM optimized with mutation SCA-PSO,
Entropy 21.4 (2019): 404.

[26] Yongbo Li, et al.,
A fault diagnosis scheme for planetary gearboxes using modified multi-scale symbolic dynamic entropy and mRMR feature selection,
Mechanical Systems and Signal Processing 91 (2017): 295-312.

[27] Jian Wang, et al.,
Fault feature extraction for multiple electrical faults of aviation electro-mechanical actuator based on symbolic dynamics entropy,
IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC), 2015.

[28] Venkatesh Rajagopalan and Asok Ray,
Symbolic time series analysis via wavelet-based partitioning,
Signal processing 86.11 (2006): 3309-3320

[29] Xiaofeng Liu, et al.,
Increment entropy as a measure of complexity for time series,
Entropy 18.1 (2016): 22.1.

[30] *** Correction on Liu, X.; Jiang, A.; Xu, N.; Xue, J. -
Increment Entropy as a Measure of Complexity for Time Series, Entropy 2016, 18, 22,
Entropy 18.4 (2016): 133.

[31] Xiaofeng Liu, et al.,
Appropriate use of the increment entropy for electrophysiological time series,
Computers in biology and medicine 95 (2018): 13-23.

[32] Theerasak Chanwimalueang and Danilo Mandic,
Cosine similarity entropy: Self-correlation-based complexity analysis of dynamical systems,
Entropy 19.12 (2017): 652.

[33] Ashish Rohila and Ambalika Sharma,
Phase entropy: a new complexity measure for heart rate variability,
Physiological measurement 40.10 (2019): 105006.

[34] David Cuesta-Frau,
Slope Entropy: A New Time Series Complexity Estimator Based on Both Symbolic Patterns and Amplitude Information,
Entropy 21.12 (2019): 1167.

[35] George Manis, M.D. Aktaruzzaman and Roberto Sassi,
Bubble entropy: An entropy almost free of parameters,
IEEE Transactions on Biomedical Engineering 64.11 (2017): 2711-2718.

[36] Chang Yan, et al.,
Novel gridded descriptors of poincaré plot for analyzing heartbeat interval time-series,
Computers in biology and medicine 109 (2019): 280-289.

[37] Chang Yan, et al.,
Area asymmetry of heart rate variability signal,
Biomedical engineering online 16.1 (2017): 1-14.

[38] Alberto Porta, et al.,
Temporal asymmetries of short-term heart period variability are linked to autonomic regulation,
American Journal of Physiology-Regulatory, Integrative and Comparative Physiology 295.2 (2008): R550-R557.

[39] C.K. Karmakar, A.H. Khandoker and M. Palaniswami,
Phase asymmetry of heart rate variability signal,
Physiological measurement 36.2 (2015): 303.

[40] Przemyslaw Guzik, et al.,
Heart rate asymmetry by Poincaré plots of RR intervals,
Biomedizinische Technik. Biomedical engineering 51.4 (2006): 272-275.

[41] Chang Francis Hsu, et al.,
Entropy of entropy: Measurement of dynamical complexity for biological systems,
Entropy 19.10 (2017): 550.

[42] Jiawei Yang, et al.,
Classification of Interbeat Interval Time-series Using Attention Entropy,
IEEE Transactions on Affective Computing (2020)

[43] X. Wang, S. Si and Y. Li,
Multiscale Diversity Entropy: A Novel Dynamical Measure for Fault Diagnosis of Rotating Machinery,
IEEE Transactions on Industrial Informatics, vol. 17, no. 8, pp. 5419-5429, 2021

[44] Y. Wang, et al.,
Cumulative Diversity Pattern Entropy (CDEn): A High-Performance, Almost-Parameter-Free Complexity Estimator for Nonstationary Time Series,
IEEE Transactions on Industrial Informatics, vol. 19, no. 9, pp. 9642-9653, 2023

[45] Omidvarnia, Amir, et al.,
Range entropy: A bridge between signal complexity and self-similarity,
Entropy 20.12 (2018): 962.

[46] Hong-Bo Xie, et al.,
Cross-fuzzy entropy: A new method to test pattern synchrony of bivariate time series,
Information Sciences 180.9 (2010): 1715-1724.

[47] Wenbin Shi, Pengjian Shang, and Aijing Lin,
The coupling analysis of stock market indices based on cross-permutation entropy, Nonlinear Dynamics 79.4 (2015): 2439-2447.

[48] Madalena Costa, Ary Goldberger, and C-K. Peng,
Multiscale entropy analysis of complex physiologic time series,
Physical review letters 89.6 (2002): 068102.

[49] Vadim V. Nikulin, and Tom Brismar,
Comment on "Multiscale entropy analysis of complex physiologic time series",
Physical review letters 92.8 (2004): 089803.

[50] Madalena Costa, Ary L. Goldberger, and C-K. Peng.
Costa, Goldberger, and Peng reply,
Physical Review Letters 92.8 (2004): 089804.

[51] Madalena Costa, Ary L. Goldberger and C-K. Peng,
Multiscale entropy analysis of biological signals,
Physical Review E 71.2 (2005): 021906

[52] Ranjit A. Thuraisingham and Georg A. Gottwald,
On multiscale entropy analysis for physiological data,
Physica A: Statistical Mechanics and its Applications 366 (2006): 323-332.

[53] Meng Hu and Hualou Liang,
Intrinsic mode entropy based on multivariate empirical mode decomposition and its application to neural data analysis,
Cognitive neurodynamics 5.3 (2011): 277-284.

[54] Anne Humeau-Heurtier
The multiscale entropy algorithm and its variants: A review,
Entropy 17.5 (2015): 3110-3123.

[55] Jianbo Gao, et al.,
Multiscale entropy analysis of biological signals: a fundamental bi-scaling law,
Frontiers in computational neuroscience 9 (2015): 64.

[56] Paolo Castiglioni, et al.,
Multiscale Sample Entropy of cardiovascular signals: Does the choice between fixed-or varying-tolerance among scales influence its evaluation and interpretation?,
Entropy 19.11 (2017): 590.

[57] Tuan D Pham,
Time-shift multiscale entropy analysis of physiological signals,
Entropy 19.6 (2017): 257.

[58] Hamed Azami and Javier Escudero,
Coarse-graining approaches in univariate multiscale sample and dispersion entropy,
Entropy 20.2 (2018): 138.

[59] Magdalena Costa and Ary Goldberger,
Generalized multiscale entropy analysis: Application to quantifying the complex volatility of human heartbeat time series,
Entropy, 17 (2015): 1197–1203

[60] Shuen-De Wu, et al.,
Time series analysis using composite multiscale entropy,
Entropy 15.3 (2013): 1069-1084.

[61] Shuen-De Wu, et al.,
Analysis of complex time series using refined composite multiscale entropy,
Physics Letters A 378.20 (2014): 1369-1374.

[62] Hamed Azami, Alberto Fernández, and Javier Escudero.,
Refined multiscale fuzzy entropy based on standard deviation for biomedical signal analysis.
Medical & biological engineering & computing 55 (2017): 2037-2052.

[63] José Fernando Valencia, et al.,
Refined multiscale entropy: Application to 24-h holter recordings of heart period variability in healthy and aortic stenosis subjects,
IEEE Transactions on Biomedical Engineering 56.9 (2009): 2202-2213.

[64] Puneeta Marwaha and Ramesh Kumar Sunkaria,
Optimal selection of threshold value 'r' for refined multiscale entropy,
Cardiovascular engineering and technology 6.4 (2015): 557-576.

[65] Ying Jiang, C-K. Peng and Yuesheng Xu,
Hierarchical entropy analysis for biological signals,
Journal of Computational and Applied Mathematics 236.5 (2011): 728-742.

[66] Antoine Jamin, et al,
A novel multiscale cross-entropy method applied to navigation data acquired with a bike simulator,
41st annual international conference of the IEEE EMBC, 2019.

[67] Antoine Jamin and Anne Humeau-Heurtier,
(Multiscale) Cross-Entropy Methods: A Review,
Entropy 22.1 (2020): 45.

[68] Rui Yan, Zhuo Yang, and Tao Zhang,
Multiscale cross entropy: a novel algorithm for analyzing two time series,
5th International Conference on Natural Computation, Vol. 1, pp: 411-413 IEEE, 2009.

[69] Yi Yin, Pengjian Shang, and Guochen Feng,
Modified multiscale cross-sample entropy for complex time series,
Applied Mathematics and Computation 289 (2016): 98-110.

[70] Luiz Eduardo Virgili Silva, et al.,
Two-dimensional sample entropy: Assessing image texture through irregularity, Biomedical Physics & Engineering Express 2.4 (2016): 045002.

[71] Luiz Fernando Segato Dos Santos, et al.,
Multidimensional and fuzzy sample entropy (SampEnMF) for quantifying H & E histological images of colorectal cancer,
Computers in biology and medicine 103 (2018): 148-160.

[72] Mirvana Hilal and Anne Humeau-Heurtier,
Bidimensional fuzzy entropy: Principle analysis and biomedical applications,
41st Annual International Conference of the IEEE (EMBC) Society 2019.

[73] Hamed Azami, Javier Escudero and Anne Humeau-Heurtier,
Bidimensional distribution entropy to analyze the irregularity of small-sized textures,
IEEE Signal Processing Letters 24.9 (2017): 1338-1342.

[74] Hamed Azami, et al.,
Two-dimensional dispersion entropy: An information-theoretic method for irregularity analysis of images,
Signal Processing: Image Communication, 75 (2019): 178-187.

[75] Haroldo Ribeiro et al.,
Complexity-Entropy Causality Plane as a Complexity Measure for Two-Dimensional Patterns,
PLoS ONE (2012), 7(8):e40689.

[76] Luciano Zunino and Haroldo Ribeiro
Discriminating image textures with the multiscale two-dimensional complexity-entropy causality plane,
Chaos, Solitons and Fractals, 91:679-688 (2016).

[77] Ricardo Espinosa, et al.,
Two-dimensional EspEn: A New Approach to Analyze Image Texture by Irregularity,
Entropy, 23:1261 (2021)

[78] Xiao, H.; Chanwimalueang, T.; Mandic, D.P.,
Multivariate Multiscale Cosine Similarity Entropy and Its Application to Examine Circularity Properties in Division Algebras,
Entropy 2022, 24, 1287.

[79] Mosabber Uddin Ahmed, Danilo P. Mandic, *Multivariate multiscale entropy: A tool for complexity analysis of multichannel data.*,
Physical Review E 84.6 (2011): 061918.

[80] Ahmed Mosabber Uddin, Danilo P. Mandic, *Multivariate multiscale entropy analysis*,
IEEE signal processing letters 19.2 (2011): 91-94.

[81] Ahmed, Mosabber U., et al.
A multivariate multiscale fuzzy entropy algorithm with application to uterine EMG complexity analysis.
Entropy 19.1 (2016): 2.

[82] H Azami, A Fernández, J Escudero,
Multivariate Multiscale Dispersion Entropy of Biomedical Times Series,
Entropy 2019, 21, 913.

[83] Matthew W. Flood,
EntropyHub: An Open-Source Toolkit for Entropic Time Series Aalysis,
PLoS One 16(11):e0259448 (2021). www.EntropyHub.xyz

6

Glossary of Function Syntax

`Bins` Method for determining the optimum number of histogram bins
- [DistEn / XDistEn / DistEn2D]

`c` Number of desired symbols in the symbolic sequence
- [CondEn / DispEn / MvDispEn / XCondEn / DispEn2D]

`Data` Matrix of N rows (> 10) and M columns (> 1) representing the M sequences of a multivariate dataset.
- [MvSampEn / MvDispEn / MvFuzzEn / MvPermEn / MvCoSiEn]

`EnType` Type of *Base*, *Cross-* or *Multivariate* entropy method to use for multiscale entropy analysis
- [MSobject]

`F_Num` Numerator of Butterworth low-pass filter cutoff frequency where the cutoff frequency at each scale (τ) becomes: $F_c = \frac{F_{Num}}{\tau}$
- [rMSEn / rXMSEn]

`F_Order` Butterworth low-pass filter order
- [rMSEn / rXMSEn]

`Fluct` Option to return fluctuation-based dispersion entropy
- [DispEn]

`Freqs` Edge frequencies of the frequency band when estimating BandEn
- [SpecEn]

Fx The type of fuzzy membership function
- [FuzzEn / XFuzzEn / MvFuzzEn / FuzzEn2D]

K The number of angular partitions in the second-order difference plot
- [PhasEn]

Lock Option to lock/unlock the matrix size when estimating bidimensional entropies
- [SampEn2D / DistEn2D / DispEn2D / FuzzEn2D]

Logx The base of the logarithm in the entropy formula
- [All functions]

Lvls Angular threshold levels
- [SlopEn]

m Embedding dimension value
- [ApEn / SampEn / FuzzEn / K2En / PermEn / CondEn / DistEn / DispEn / SyDyEn / IncrEn / CoSiEn / SlopEn / BubbEn / RangEn / DivEn / XApEn / XSampEn / XFuzzEn / XCondEn / XK2En / XPermEn / XDistEn]
Vector of embedding dimension values
- [MvSampEn / MvFuzzEn / MvDispEn / MvPermEn / MvCoSiEn]
Number of grid partitions
- [GridEn]
Size of sub-matrix for bidimensional entropy estimation
- [SampEn2D / FuzzEn2D / DistEn2D / DispEn2D / EspEn2D]

Mat Image (matrix) for estimating bidimensional entropies
- [SampEn2D / DistEn2D / DispEn2D / FuzzEn2D / EspEn2D]

Methodx Time series graining method for multiscale entropy analysis
- [MSEn / XMSEn / MvMSEn]
Option for multivariate dispersion entropy algorithm
- [MvDispEn]

N Number of signal points to use for fast Fourier transform. If **N** is shorter than the length of **Sig**, only the first **N** points are used. When **N** is greater than the length of **Sig**, the signal is zero-padded to length **N**
- [SpecEn / XSpecEn]

Norm Normalization of the returned entropy value (method dependent)
- [PermEn / CondEn / DistEn / SpecEn / DispEn / SyDyEn / CoSiEn / PhasEn / IncrEn / SlopEn / XCondEn /XDistEn / XSpecEn / DistEn2D / DispEn2D / PermEn2D / MvPermEn / MvDispEn]
Normalization of input data
- [MvCoSiEn / MvSampEn / MvFuzzEn]

Plotx Option to return a plot with the entropy value (method dependent)
- [PhasEn / GridEn / MSEn / XMSEn / cMSEn / rMSEn / cXMSEn / rXMSEn / hMSEn / hXMSEn / MvMSEn / cMvMSEn]

ps Percentage similarity
- [EspEn2d]

r Distance threshold for matching vectors
- [ApEn / SampEn / MvSampEn / XApEn / XSampEn / SampEn2D / EspEn2D]
Angular threshold for matching vectors
- [CoSiEn / MvCoSiEn]
Parameters for the fuzzy membership function
- [FuzzEn / FuzzEn2D / XFuzzEn / MvFuzzEn]

R Quantifying resolution
- [IncrEn]

RadNew Option to rescale the distance matching threshold at each temporal scale for multiscale entropy analysis
- [MSEn / cMSEn / rMSEn / hMSEn / XMSEn / cXMSEn / rXMSEn / hXMSEn]

Refined Option to perform refined-composite multiscale entropy
- [cMSEn / cXMSEn / cMvMSEn]

rho Tuning parameter for estimating fine-sorted dispersion entropy
- [DispEn]

S Number of slices (s1, s2)
- [EnofEn]

Scales Number of temporal scales for performing multiscale entropy analysis
- [All multiscale methods]

Sig Time series signal(s)
- [All *Base* and *Cross*-entropy methods]

tau Time delay
- [All *Base* and *Cross*-entropy methods]

Typex Type of permutation entropy method
- [PermEn / MvPermEn]
Type of symbolic sequence transform method
- [DispEn / SyDyEn / DispEn2D / MvDispEn]

tpx Tuning parameter for the permutation entropy method specified by **Typex**
- [PermEn / MvPermEn]

Vcp Option to estimate the variance of the conditional probabilities in the sample entropy estimate.
- [SampEn, XSampEn]