THE ENTROPYHUB GUIDE

A user manual for the EntropyHub toolkit

Matthew W. Flood

www.EntropyHub.xyz

v.0.1.1 (2021)

– 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

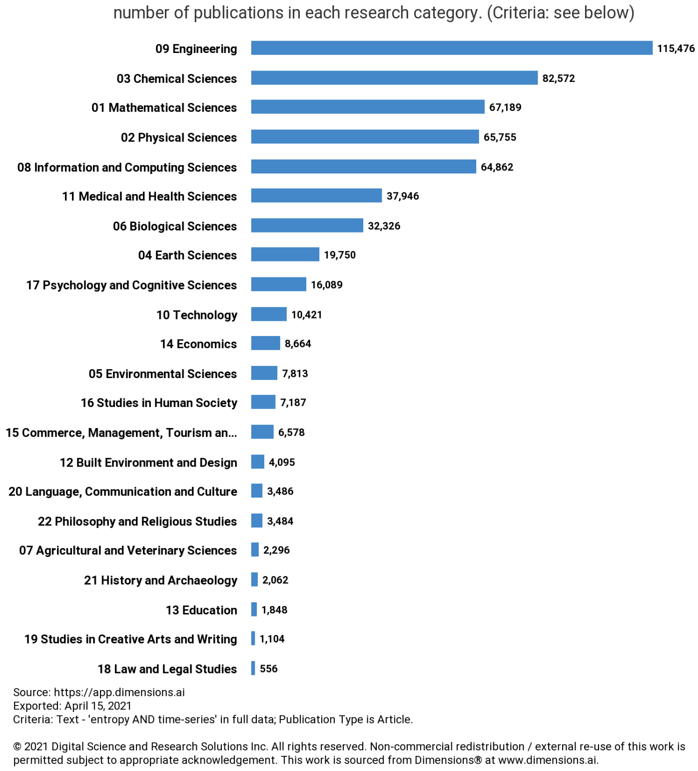
Preface

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



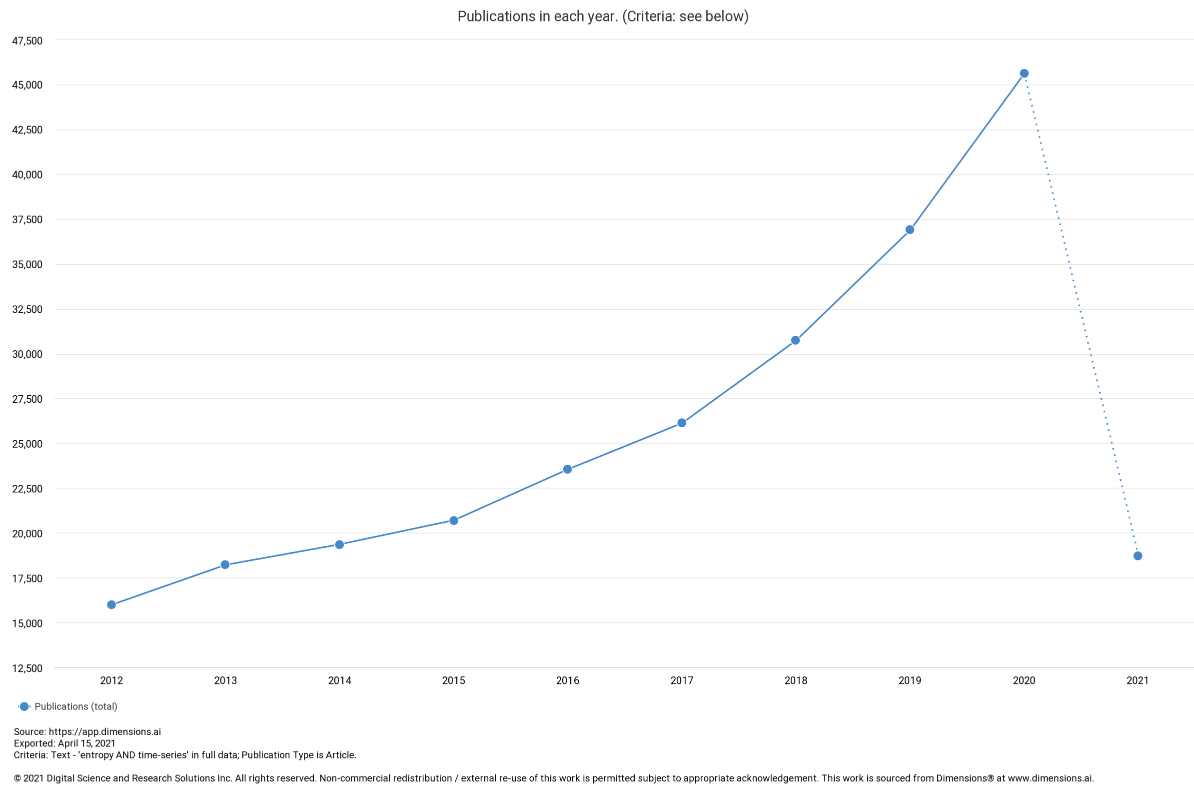


Figure 1: Research domains and the number of publications each year featuring the terms 'Entropy' AND 'Time-Series' *from 2012-2021. (Source: Dimensions.ai)*

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.

EntropyHub is licensed under the Apache License (Version 2.0) and is free to use by all on condition that the following reference be included on any outputs realized using the software:

Matthew W. Flood and Bernd Grimm (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

It is important to clarify at the outset that the term *entropy* henceforth described refers to entropy in the context of probability theory and information theory as defined by Shannon[1] , and not thermodynamic or other entropies from classical physics.

EntropyHub functions fall into five categories:

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

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

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

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 [7]. 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		
Increment Entropy	IncrEn	*Bidimensional Entropy*	*Function*
Cosine Similarity Entropy	CoSiEn	2D Sample Entropy	SampEn2D
Phase Entropy	PhasEn	2D Fuzzy Entropy	FuzzEn2D
Slope Entropy	SlopEn	2D Distribution Entropy	DistEn2D
Bubble Entropy	BubbEn	2D Dispersion Entropy	DispEn2D
Gridded Distribution Entropy	GridEn		
Entropy of Entropy	EnofEn		
Attention Entropy	AttnEn		
Multiscale Entropy	*Function*	*Multiscale Cross-Entropy*	*Function*
Multiscale Entropy	MSEn	Multiscale Cross-Entropy	XMSEn
Composite Multiscale Entropy (+ Refined-Composite Multiscale Entropy)	CMSEn	Composite Multiscale Cross-Entropy (+ Refined-Composite Multiscale Cross-Entropy)	cXMSEn
Refined Multiscale Entropy	rMSEn	Refined Multiscale Cross-Entropy	rXMSEn
Hierarchical Multiscale Entropy	hMSEn	Hierarchical Multiscale Cross-Entropy	hXMSEn

Table 1.1: List of functions in the EntropyHub toolkit.

1.1 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/0.1.0 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:

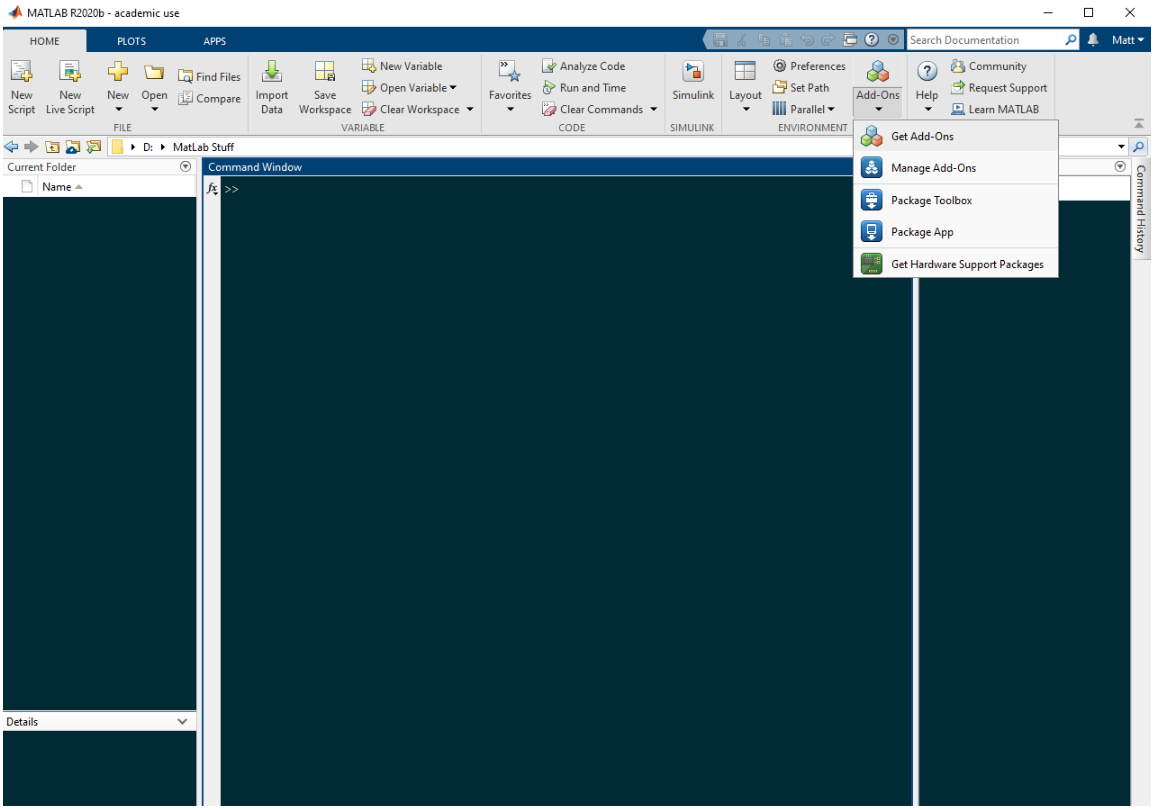
Signal Processing Toolbox

Statistics and Machine Learning Toolbox

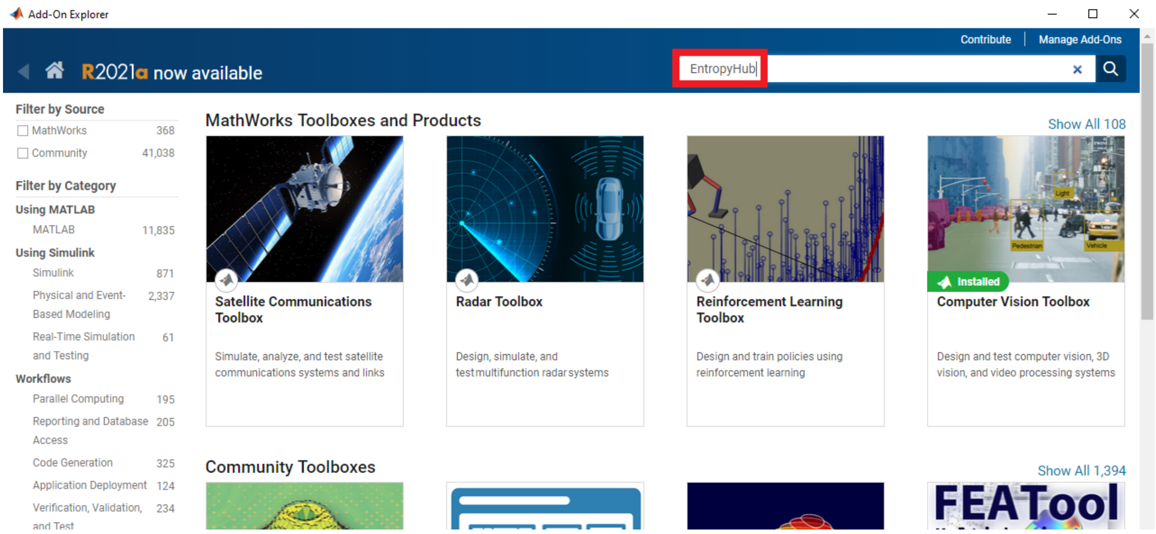
however, most functions will work without these toolboxes.
EntropyHub is intended for use with MatLab versions >= 2016a. In some cases the toolkit may work on versions 2015a and 2015b. However, it is not recommended to install on MatLab versions older than 2016 and should be done so with caution.
There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

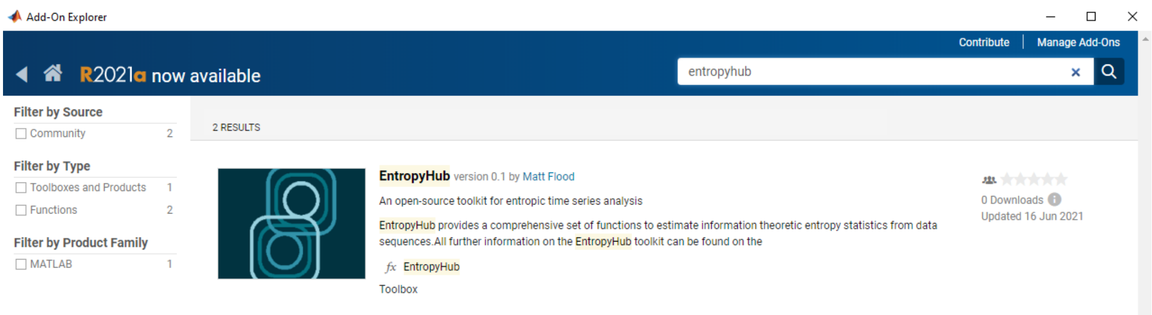
Method 1.

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

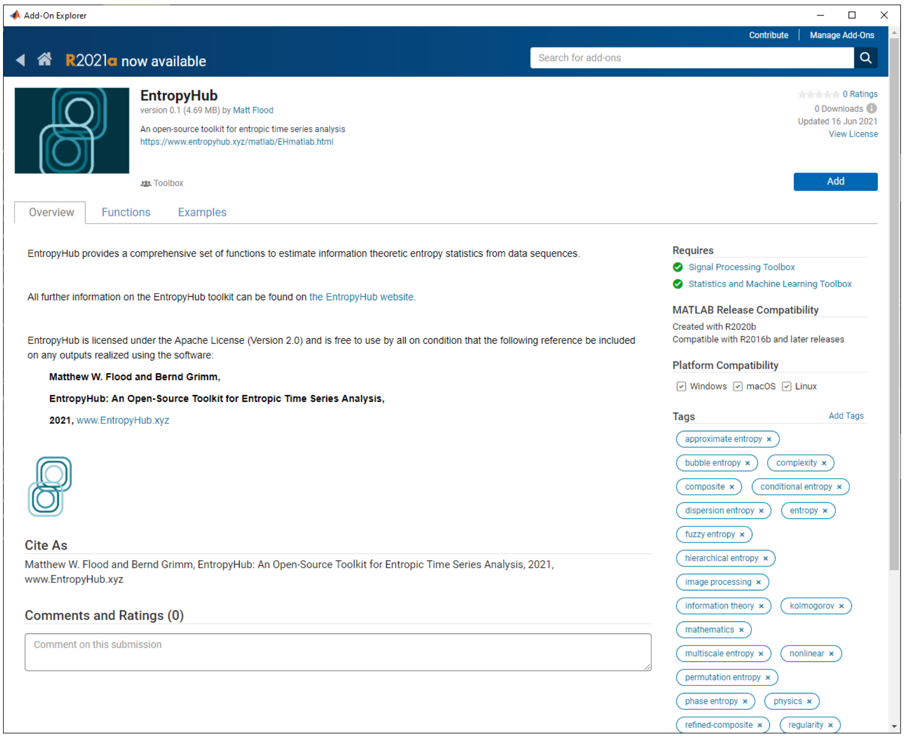


2. In the Add-On Explorer, search for ‘EntropyHub’.

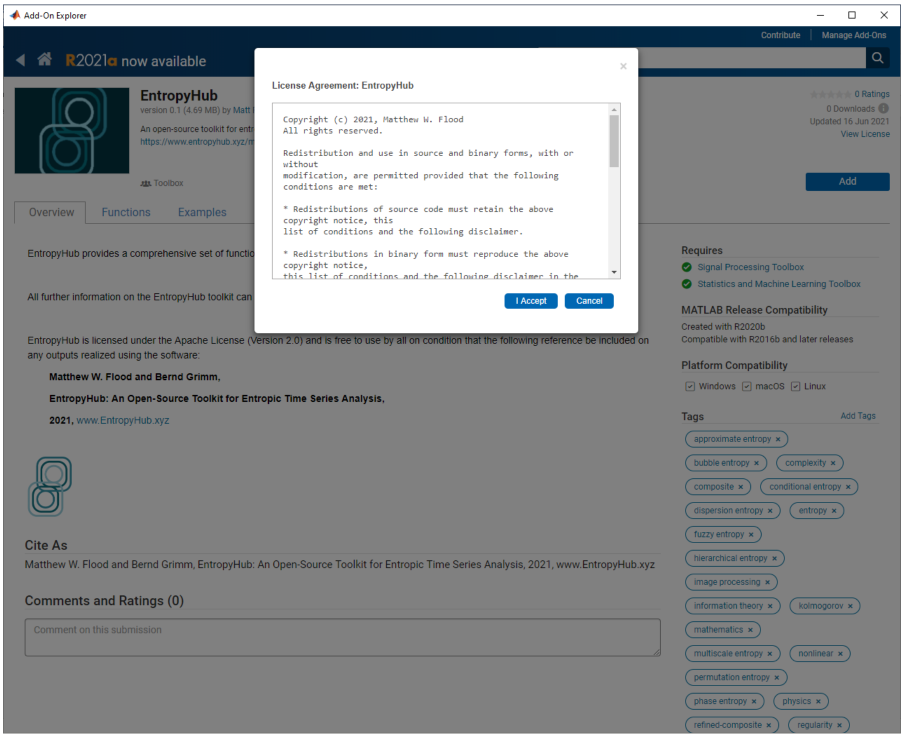




3. Open the link to EntropyHub and click the ‘Add’ button in the top right corner.

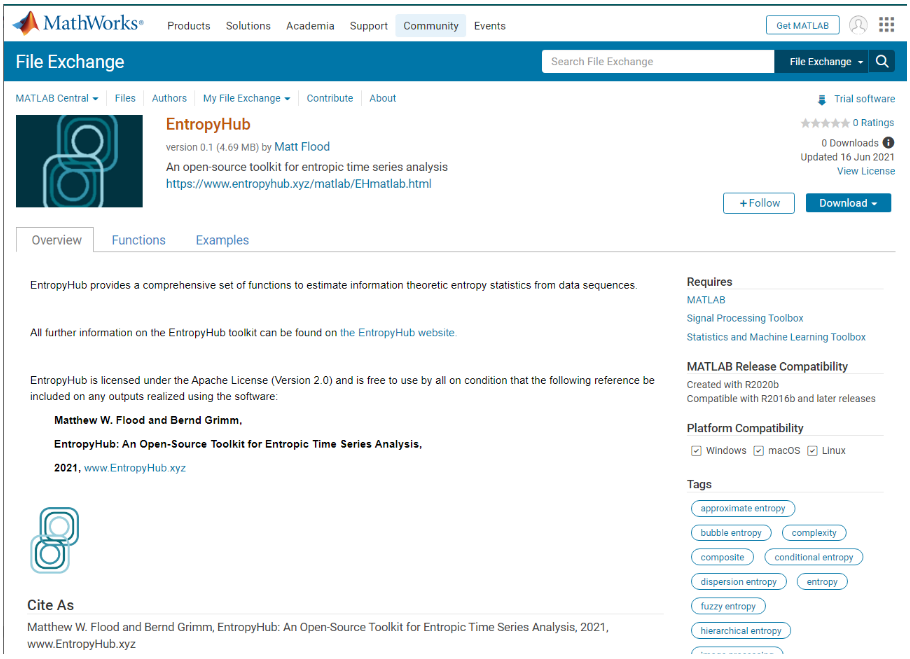


You will be asked to accept the License Agreement prior to installation.



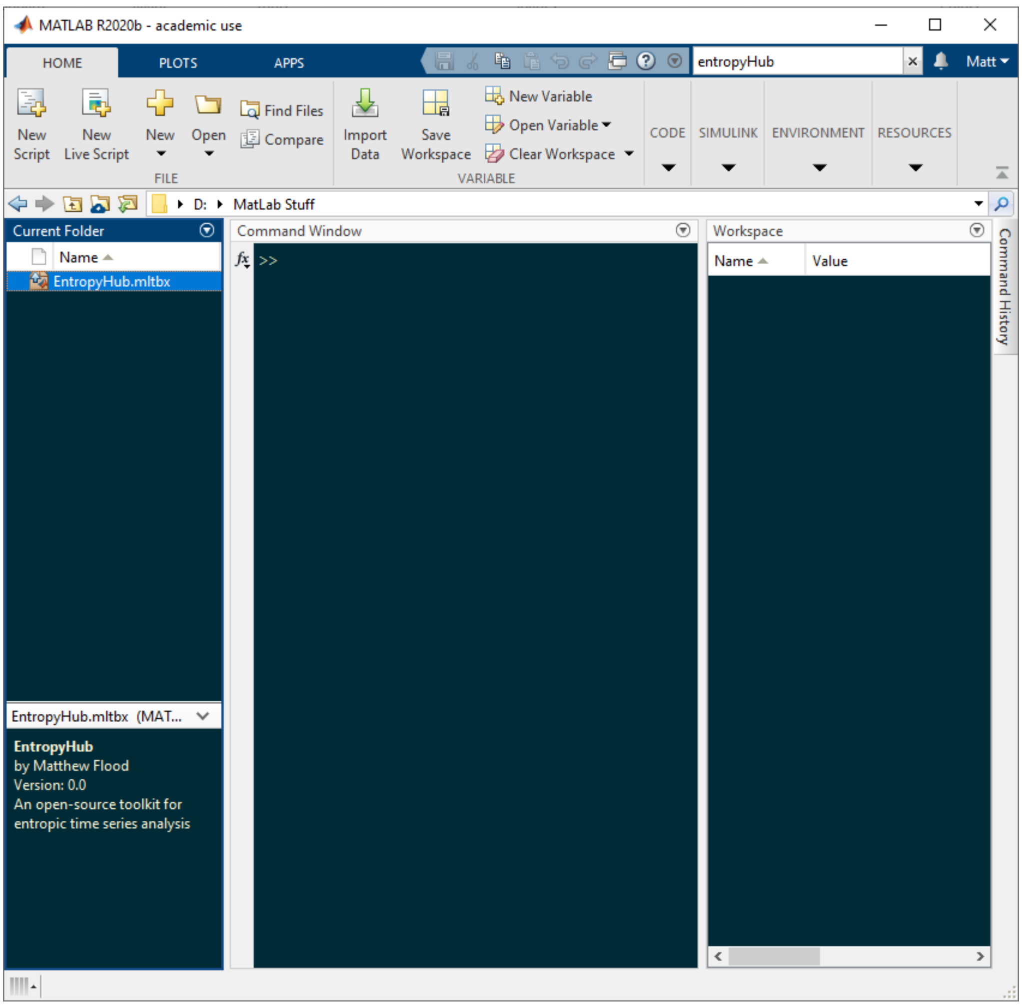
Method 2.

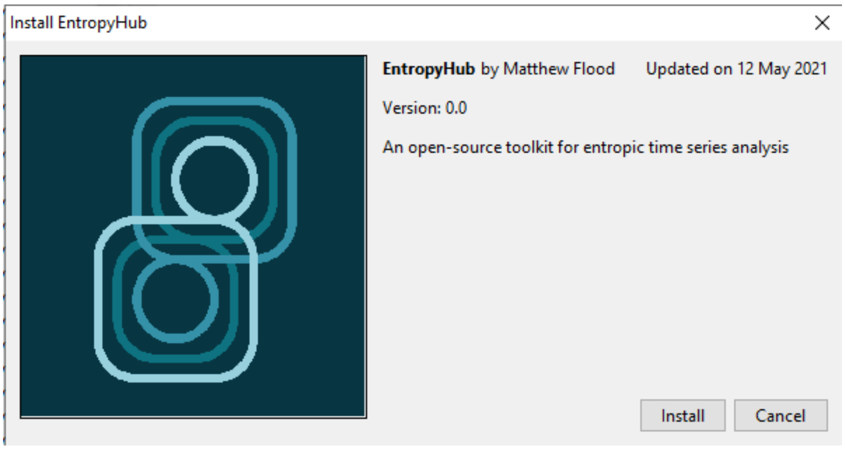
1. Visit the EntropyHub File Exchange page.
   Note: you need to be logged in to your MathWorks account to continue.



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

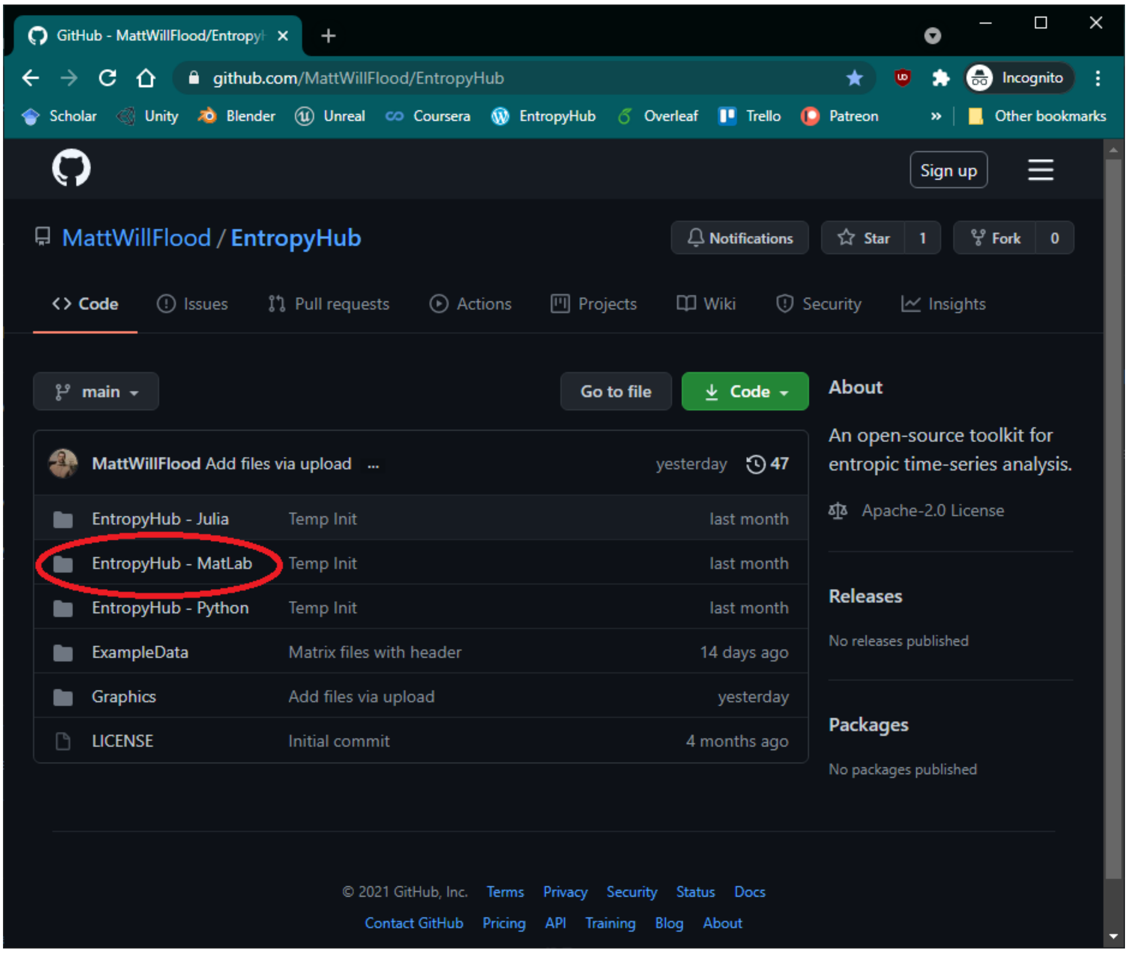
3. In MatLab, navigate the current folder to the directory where the EntropyHub.mltbx file is saved. Open the file and click install.



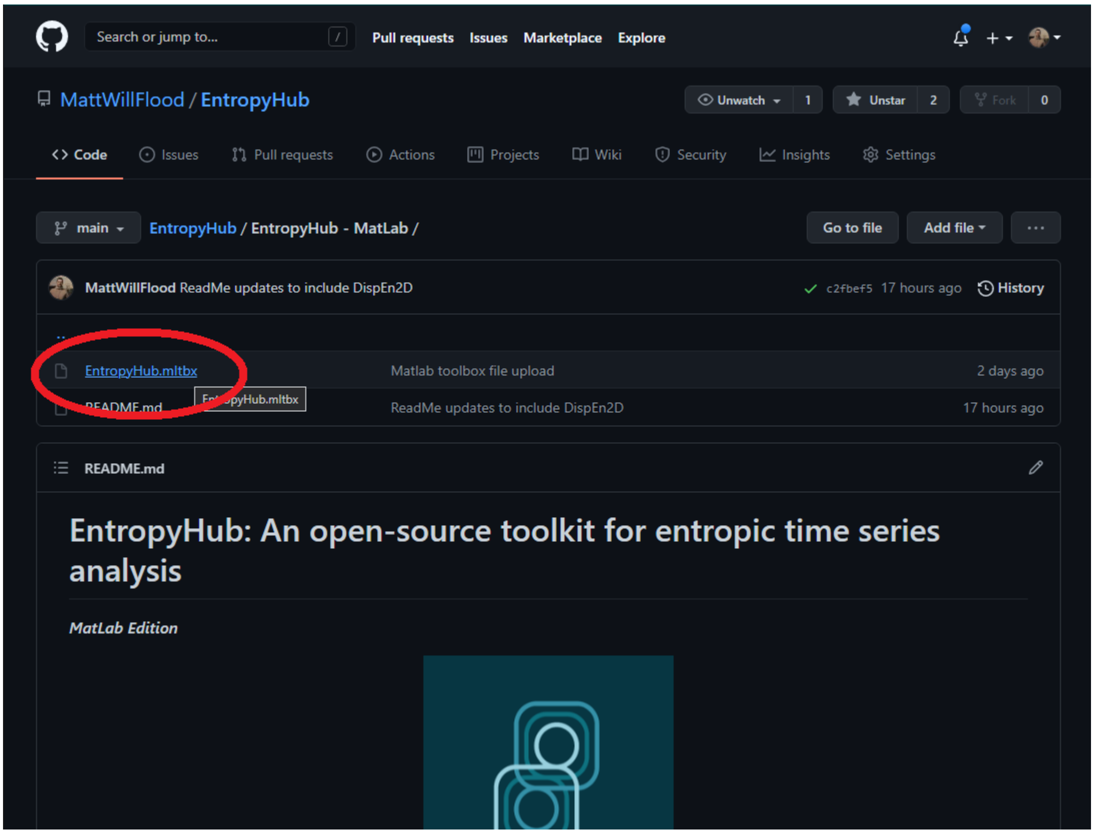


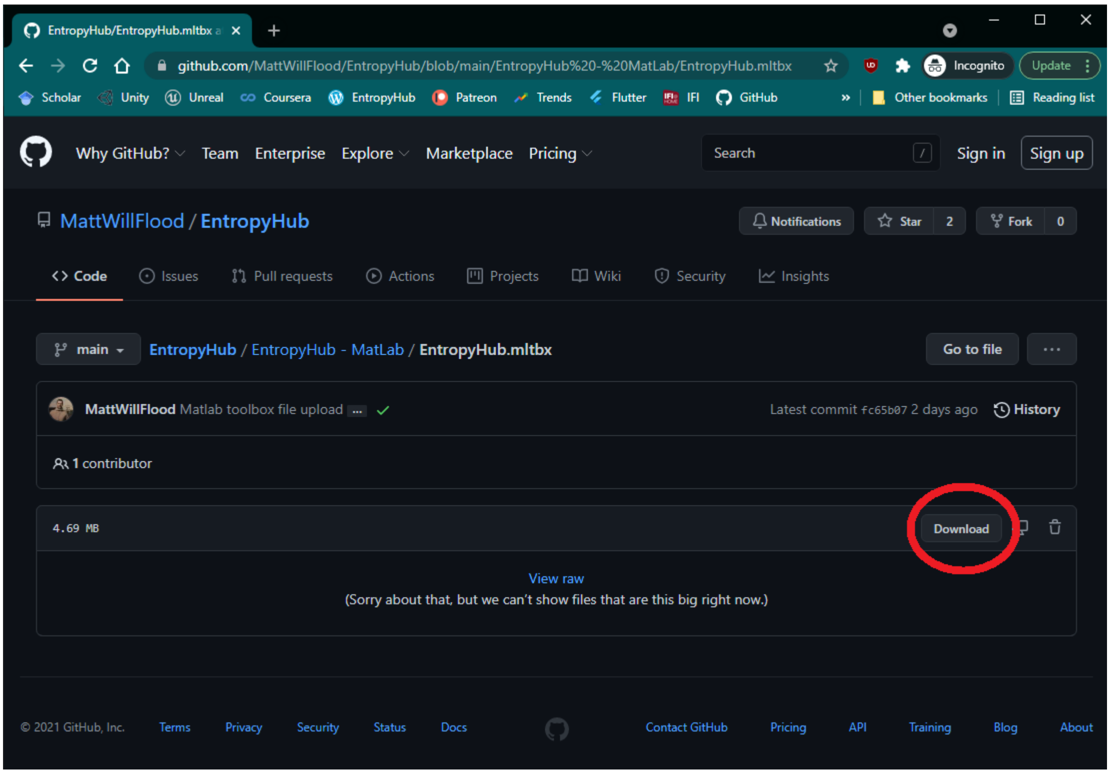
Method 3.

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

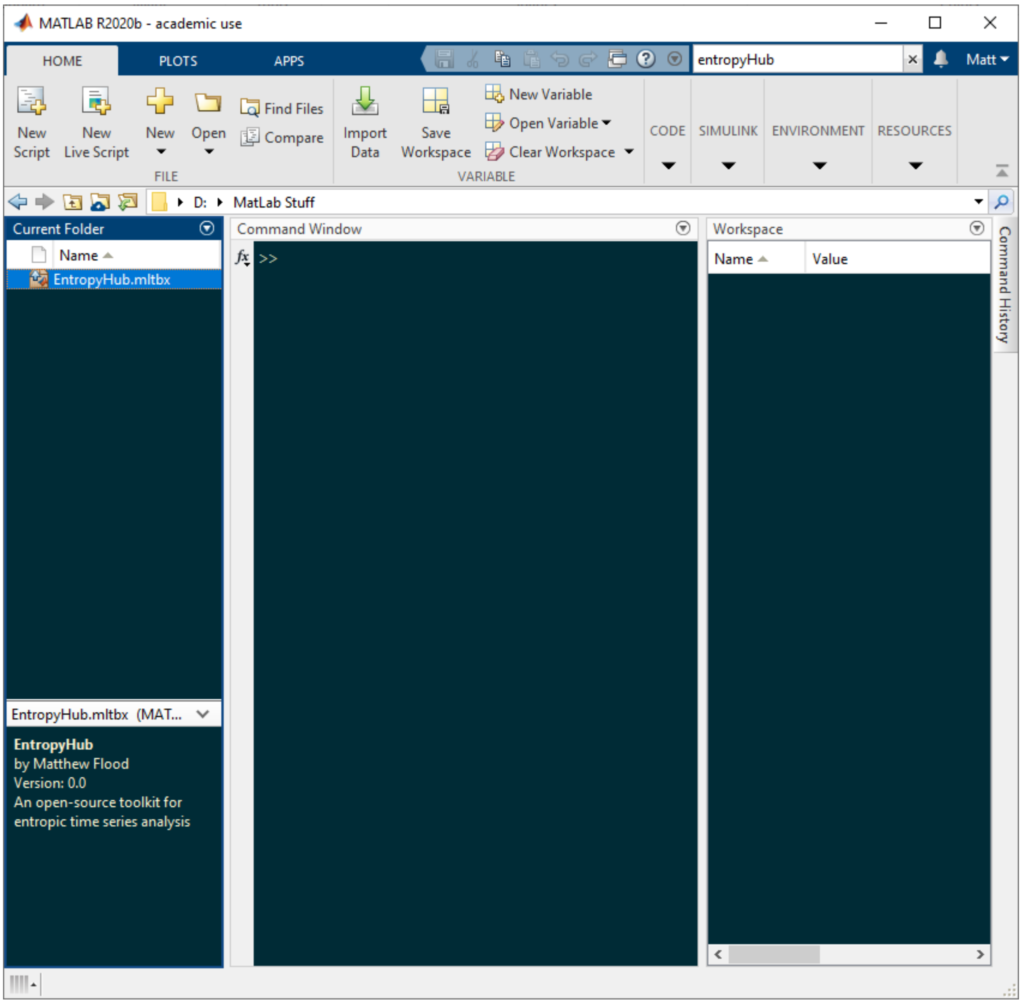


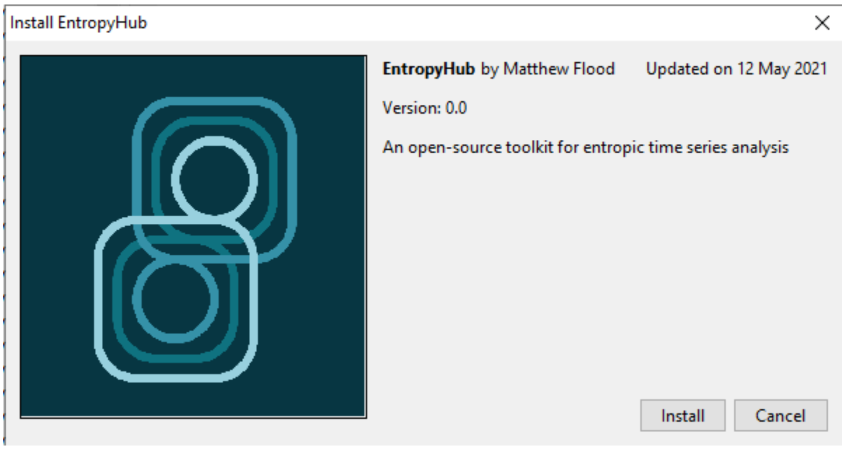
2. Open on the link to the toolbox file (EntropyHub.mltbx) and click the button labelled 'Download'.





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
FILE
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
VARIABLE
CODE
SIMULINK
ENVIRONMENT
RESOURCES
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
```

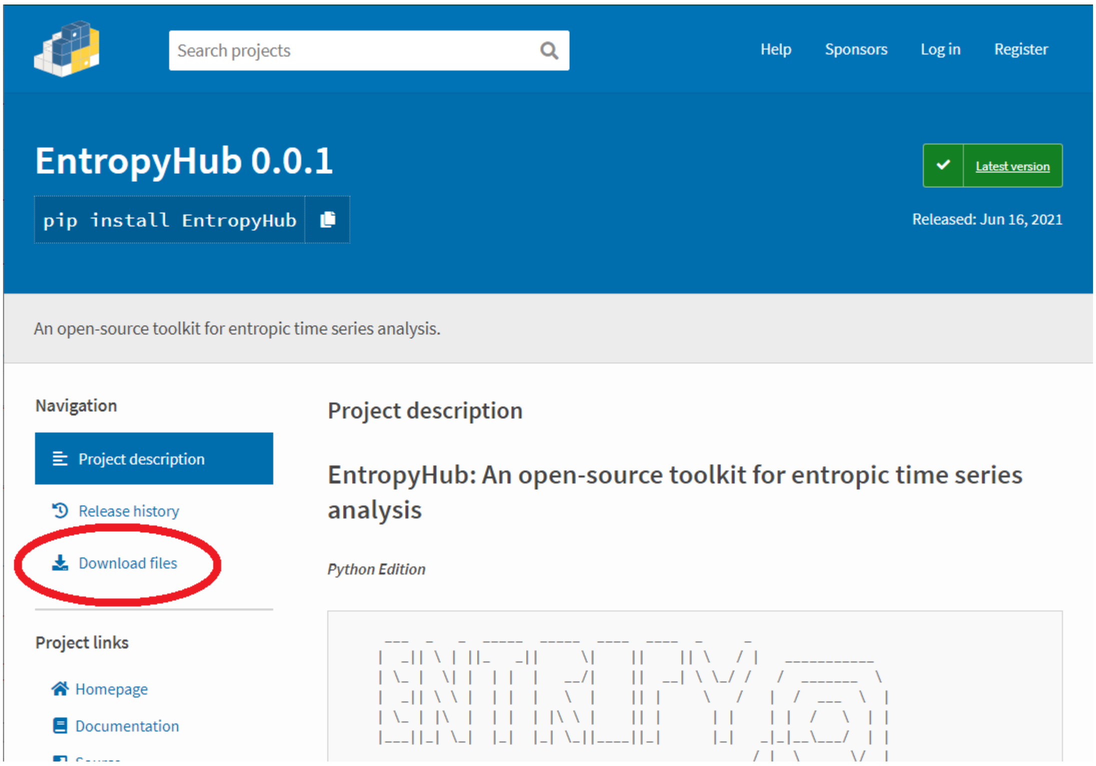
   or in abbreviated form,

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

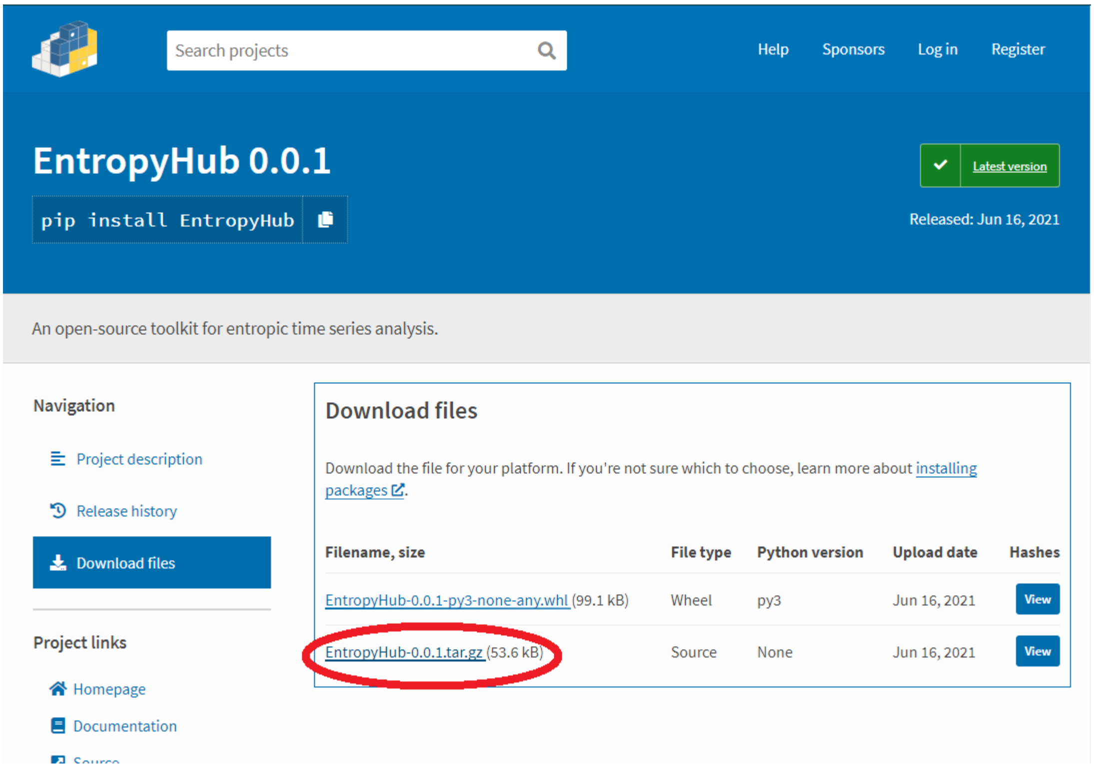
Method 2.

***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 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 – 3.5 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. $\frac{Log(N)}{N} - \Phi_1$, 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] = SampEn(Sig, 'm', 2, 'tau', 1, 'r', 0.2*std(Sig), 'Logx', exp(1))
Samp, A, B = SampEn(Sig, m = 2, tau = 1, r = 0.2*np.std(Sig), Logx = np.exp(1))
Samp, A, B = SampEn(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

Samp	Sample entropy estimates, a vector of length **m+1**. **The first value of **Samp** is the zeroth estimate, i.e. $\frac{1}{N}Log(N(N-1))–Log(A_1)$, 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.

References [2]

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

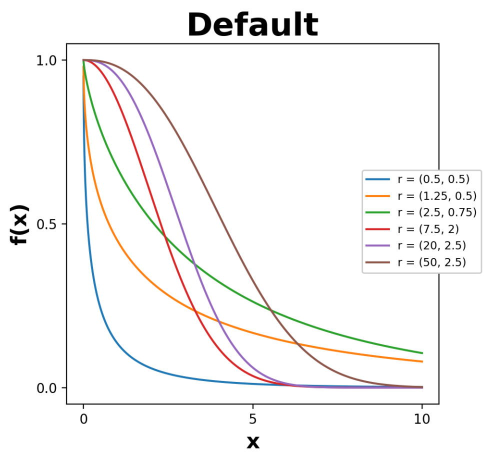
Sig Time series signal, a vector of length > 10.

m Embedding dimension, a positive integer.

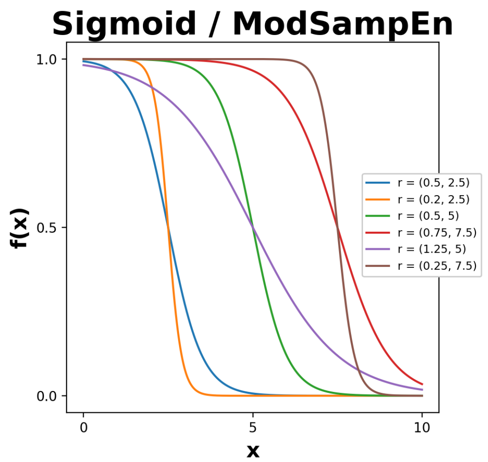
tau Time delay, a positive integer.

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

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



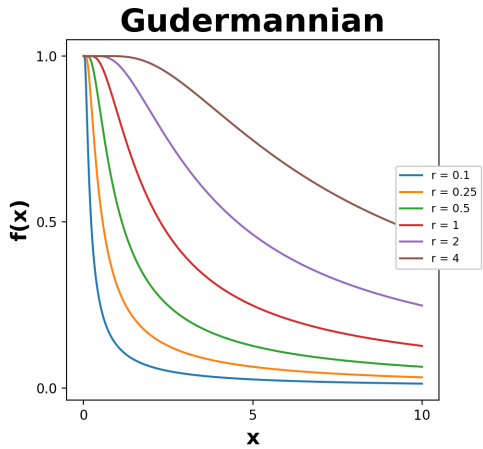
"sigmoid"/"modsampen" $f(x) = (1 + exp(\frac{x-r_2}{r_1}))^{-1}$



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

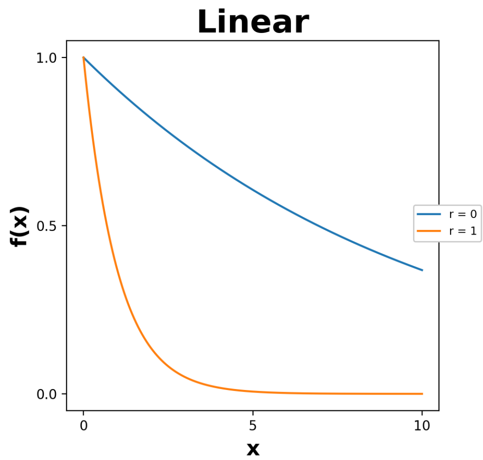
$f(x) = \frac{g(x)}{g(x_{max})}$

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



"linear"

If $\mathbf{r} = 0$:
$f(x) = exp(-\frac{x - x_{min}}{x_{max} - x_{min}})$
If $\mathbf{r} = 1$:
$f(x) = exp(-(x - x_{min}))$



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)
Gudermannian	A scalar value
Linear	0 or 1

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

[3] [4]

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 [5] [6]

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 [8] **"modified"** Modified permutation entropy [9] **"weighted"** Weighted permutation entropy [10] **"ampaware"** Amplitude-aware permutation entropy [11] **"edge"** Edge permutation entropy [12] **"uniquant"** Uniform quantization-based permutation entropy [13]
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).
Logx	Logarithm base in the entropy formula, a positive scalar.
Norm	Normalisation of **Perm** 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 [14]

References [7] [8] [9] [10] [11] [12] [13] [14]

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 [15]

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 [16]

3.1.8 SpecEn: Spectral Entropy

Syntax

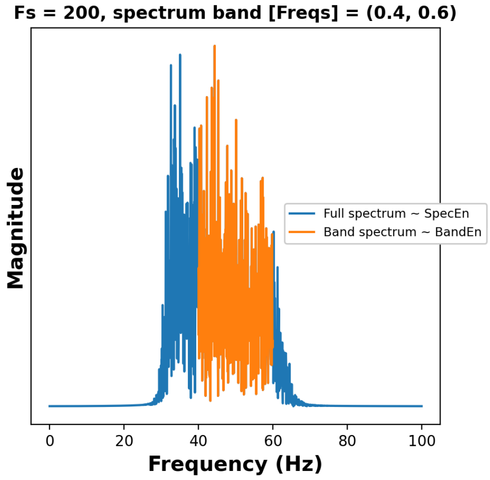
```
[Spec, BandEn] = SpecEn(Sig, 'N', 2*length(Sig)+1, 'Freqs', [0,1], 'Logx', exp(1),
'Norm', true)
Spec, BandEn = SpecEn(Sig, N = 2*len(Sig) + 1, Freqs = (0,1), Logx = np.exp(1),
Norm = True)
Spec, BandEn = SpecEn(Sig, N = 2*length(Sig) + 1, Freqs = (0,1), Logx = exp(1),
Norm = true)
```

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 [17] [18]

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.

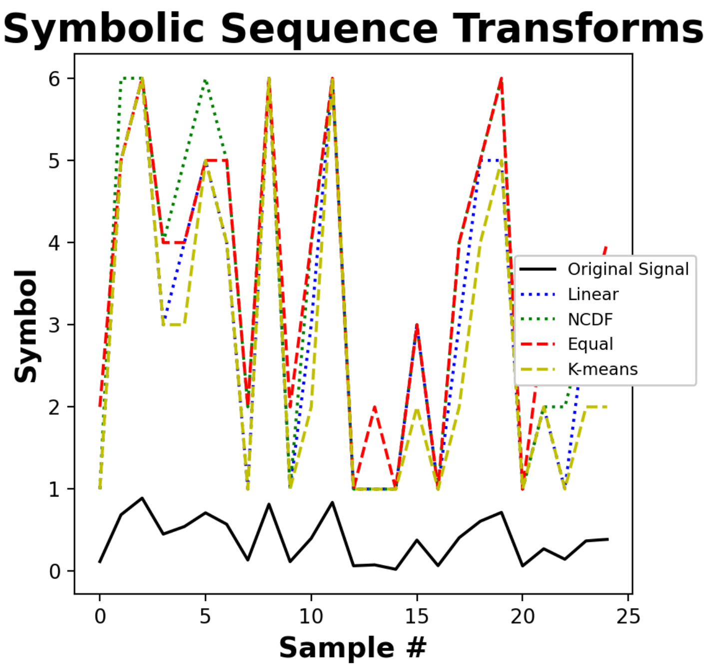
3.1.9 DispEn: Dispersion Entropy

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 [19]
	"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 [22]
	"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 [20]
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. [21]

References [19] [20] [21] [22]

3.1.10 SyDyEn: Symbolic Dynamic Entropy

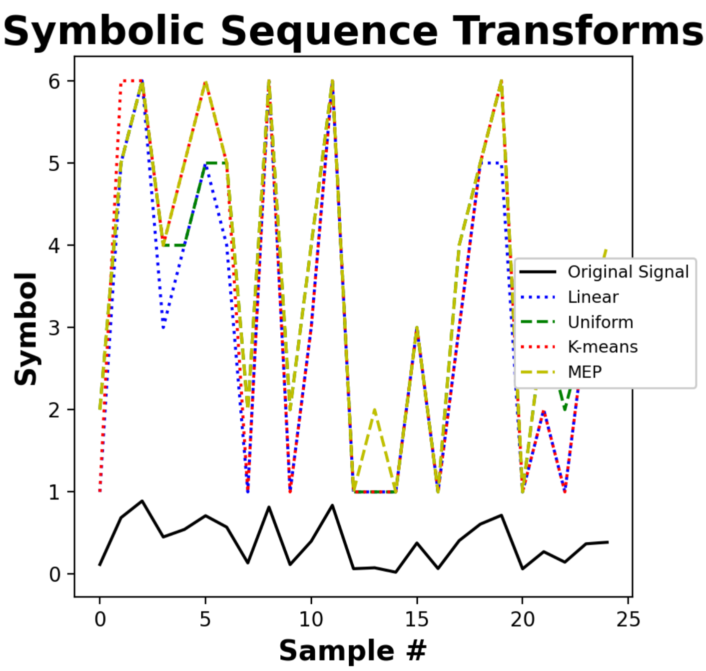
Syntax

```
[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 [24]
- **"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 [23] [24] [25]

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 [26] [27] [28]

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 interger in range [0 4]: 0 - no normalisation (default) 1 - median removed 2 - mean removed 3 - normalised by standard deviation 4 - normalised to range [-1 1]

Outputs

CoSi	Cosime similarity entropy estimate.
Bm	Global probabilities.

References [29]

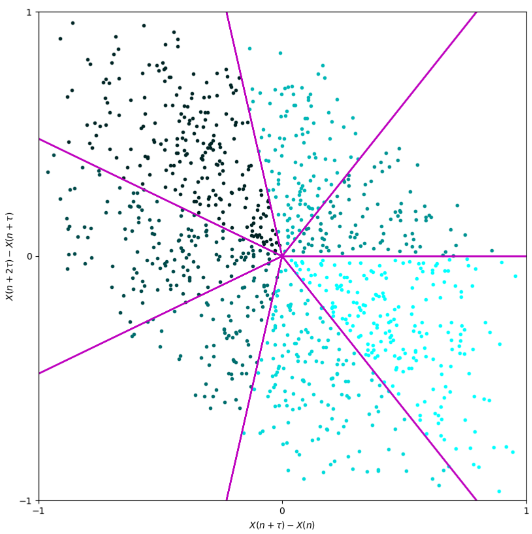
3.1.13 PhasEn: Phase Entropy

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 [30]

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 [31]

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 [32]

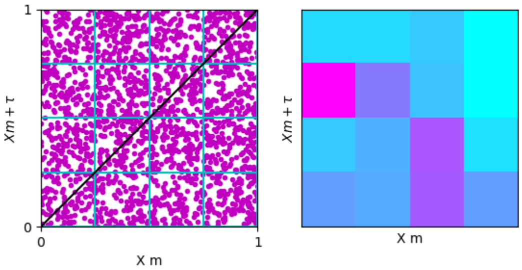
3.1.16 GridEn: Gridded Distribution Entropy

Syntax

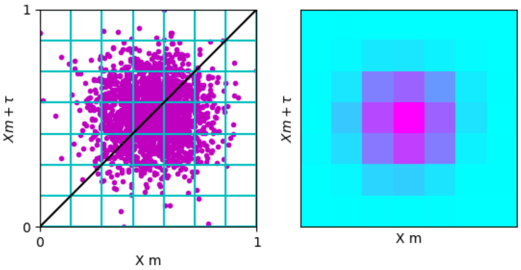
```
[GDE, GDR, PIx, GIx, SIx, AIx] = GridEn(Sig, 'm', 3, 'tau', 1, 'Logx', exp(1),
'Plotx', false)
GDE, GDR, PIx, GIx, SIx, AIx = GridEn(Sig, m = 3, tau = 1, Logx = np.exp(1),
Plotx = False)
GDE, GDR, PIx, GIx, SIx, AIx = GridEn(Sig, m = 3, tau = 1, Logx = exp(1), Plotx
= false)
```

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). [35]
GIx	Proportion of point distances above the LI. [37]
SIx	Ratio of phase angles (w.r.t. LI) of the points above the LI.[36]
AIx	Ratio of the cumulative area of sectors of points above the LI.[34]

References

[33] [34] [35] [36] [37]

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 [38]

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 [39]

3.2 Cross-Entropy Functions

3.2.1 XApEn: Cross-Approximate Entropy

Syntax

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

Arguments

Sig	Time series signals, a N x 2 matrix where N > 10. **NOTE: XApEn is direction-dependent**. Thus, the first column of **Sig** is used as the template data sequence, and the second column 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. $(Log(N)/N)$–Φ_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, Phi] = XSampEn(Sig, 'm', 2, 'tau', 1, 'r', 0.2*std(Sig), 'Logx', exp(1))
XSamp, Phi = XSampEn(Sig, m = 2, tau = 1, r = 0.2*np.std(Sig), Logx = np.exp(1))
XSamp, Phi = XSampEn(Sig, m = 2, tau = 1, r = 0.2*std(Sig), Logx = exp(1))
```

Arguments

Sig	Time series signals, a N x 2 matrix where N > 10.
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.

Outputs

XSamp	Cross-sample entropy estimates, a vector of length m+1. **The first value of **XSamp** is the zeroth estimate, i.e. $\frac{Log(N(N-1)}{N} - Log(A_1)$, 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.

References [2]

3.2.3 XFuzzEn: Cross-Fuzzy Entropy

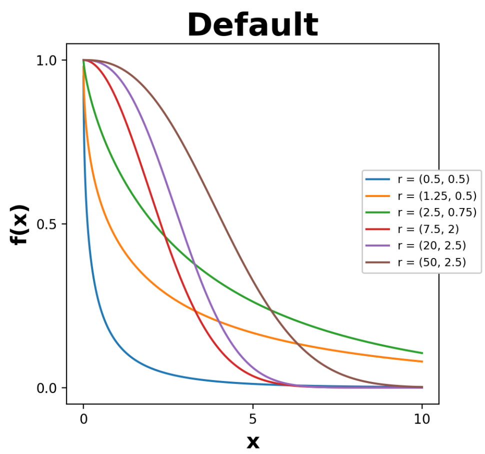
Syntax

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

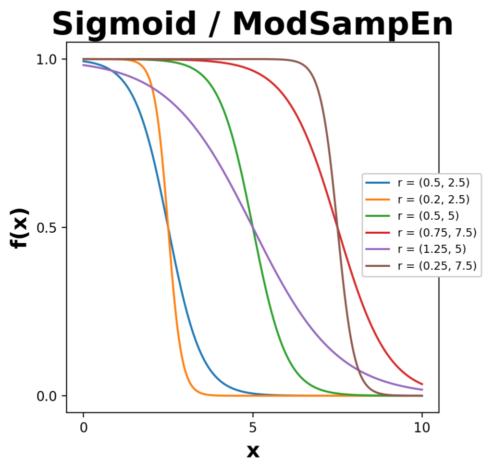
Arguments

Sig Time series signals, a N x 2 matrix where N > 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:

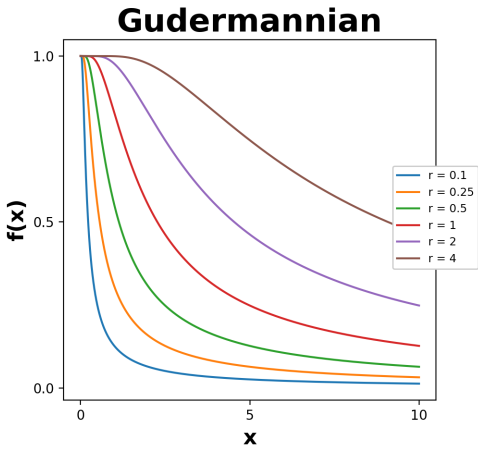
"default" $f(x) = exp(-\frac{x^{r_2}}{r_1})$



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

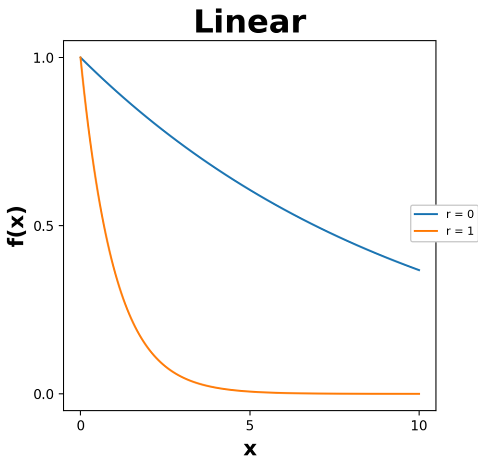


"gudermannian" $g(x) = atan(\frac{tanh(r_1)}{x})$
$f(x) = \frac{g(x)}{g(x_{max})}$
Note: Distances are normalized w.r.t. maximum distance relative to each state vector.



"linear"

If **r** $= 0$:
$f(x) = exp(-\frac{x - x_{min}}{x_{max} - x_{min}})$
If **r** $= 1$:
$f(x) = exp(-(x - x_{min}))$



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)
Gudermannian	A scalar value
Linear	0 or 1

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 [40]

3.2.4 XK2En: Cross-Kolmogorov Entropy

Syntax

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

Arguments

Sig	Time series signals, a N x 2 matrix where N > 10.
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 [67]

3.2.5 XPermEn: Cross-Permutation Entropy

Syntax

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

Arguments

Sig	Time series signals, a N x 2 matrix where N > 10.
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 [41]

3.2.6 XCondEn: Cross-Conditional Entropy

Syntax

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

Arguments

Sig	Time series signals, a N x 2 matrix where N > 10. **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 [15]

3.2.7 XDistEn: Cross-Distribution Entropy

Syntax

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

Arguments

Sig	Time series signals, a N x 2 matrix where N > 10.
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 [16]

3.2.8 XSpecEn: Cross-Spectral Entropy

Syntax

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

Arguments

Sig	Time series signals, a N x 2 matrix where N > 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==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 [67]

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 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 [42]

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

Time-shifted multiscale approximate entropy with varying tolerance across scales [43]

```
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 [53]

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

Refined-Composite multiscale entropy calculated in bits [54]

```
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)
```

3.3.1 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.

3.3.2 MSEn: Multiscale Entropy

Syntax

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

Arguments

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

Scales Number of grained time scales, a positive integer.

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

"coarse" [42]

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

"modified"

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

"timeshift" [51]

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

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

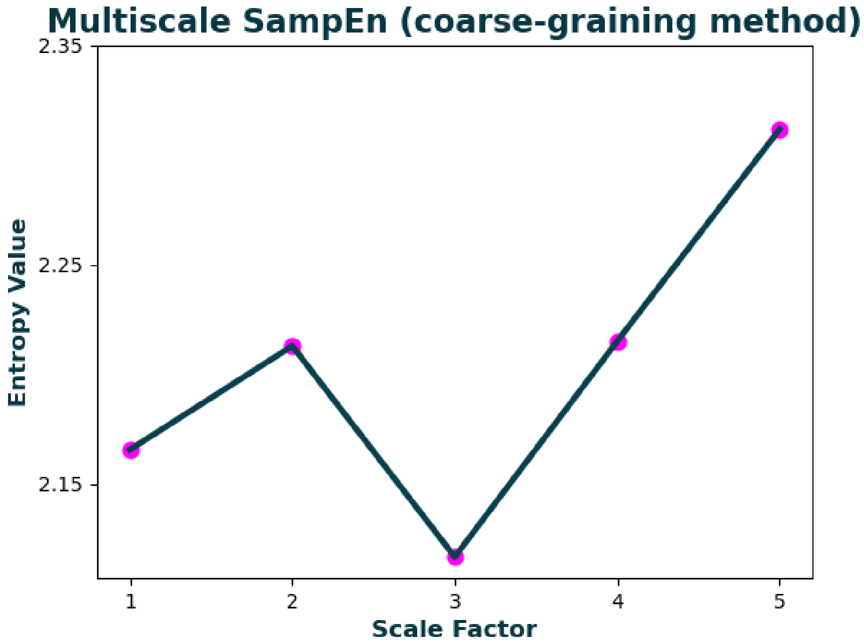
"imf" Grained time series at scale τ is the cumulative sum of intrinsic mode functions (IMF^1 to IMF^τ), where IMF^1 is the first sifting. [47]
***Note:** The empirical mode decomposition method use to derive the IMFs differs slightly between **MatLab, Python** and **Julia**, so **MSx** values will be inconsistent between the environment.
****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 [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] [52]

3.3.3 cMSEn: Composite & Refined-Composite Multiscale Entropy

Syntax

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

Arguments

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

Scales Number of time scales, a positive integer.

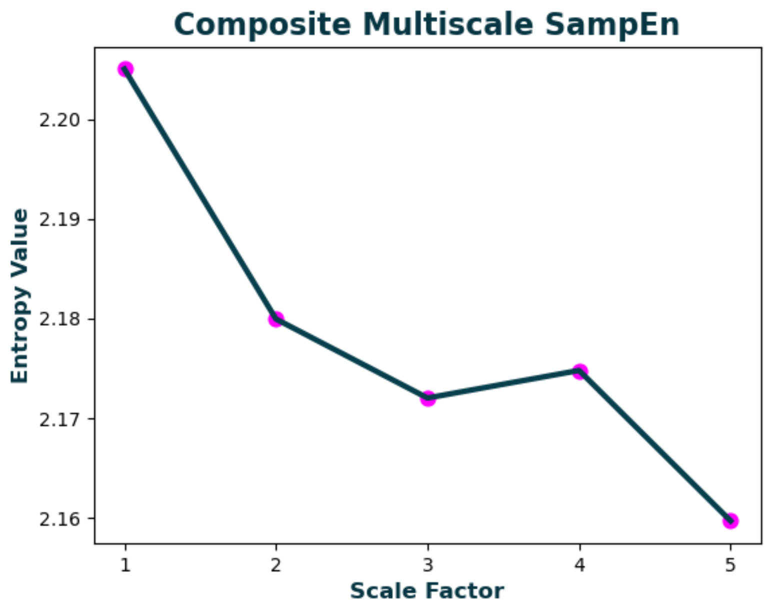
RadNew Radius rescaling method, an integer in the range [0 4].
When the **Base** entropy method specified by **Mobj** is **SampEn** or **ApEn**, **RadNew** allows the radius threshold to be updated based on the grained signal at each time scale (X_τ). If a radius threshold value **(r)** is specified 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:

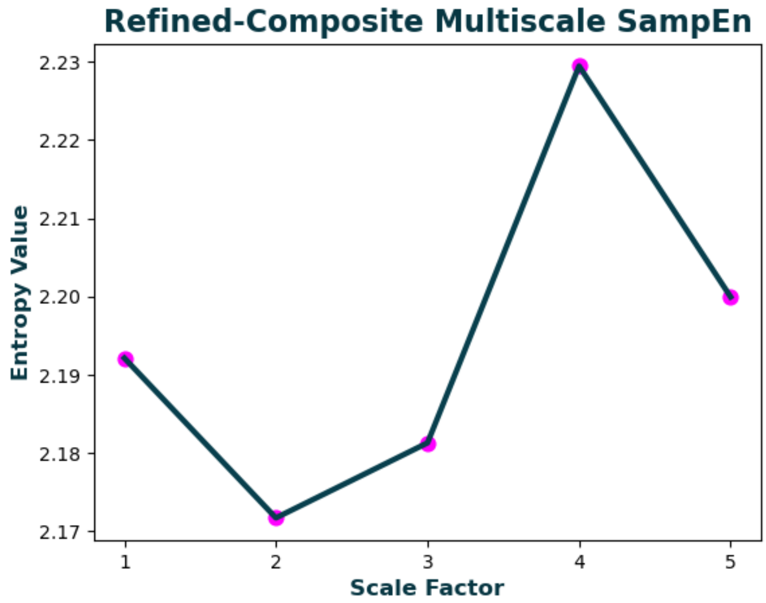
0 no rescaling
1 Standard Deviation - $r\sigma_{X_\tau}$
2 Variance - $r\sigma^2_{X_\tau}$
3 Mean Absolute Deviation - $r(\frac{1}{N}\sum|X_\tau - \bar{X_\tau}|)$
4 Median Absolute Deviation - $r(median(|X_\tau - median(X_\tau)|))$

Refined When **Refined == true** and the entropy function (**EnType**) contained in **Mobj** is **SampEn,** `cMSEn` returns the refined-composite multiscale entropy (**rcMSEn**). [54]

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.





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 [42] [43] [44] [53] [54]

3.3.4 rMSEn: Refined Multiscale Entropy

Syntax

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

Arguments

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

Scales Number of time scales, an integer > 0.

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

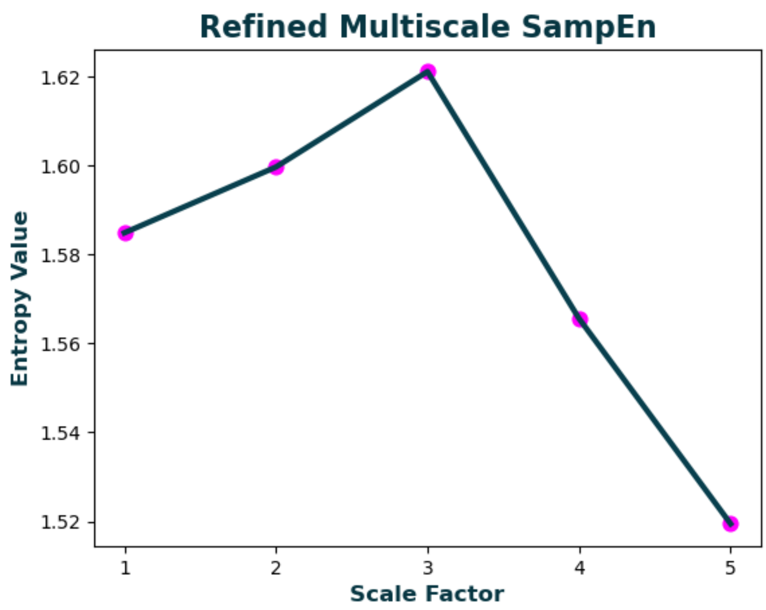
F_Num Numerator of Butterworth low-pass filter cutoff frequency, where $[0 < F_{Num} < 1]$. The cutoff frequency at each scale (τ) becomes: $F_c = \frac{F_{Num}}{\tau}$ (default: 0.5)

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 [42] [43] [44] [55] [56]

3.3.5 hMSEn: Hierarchical Multiscale Entropy

Syntax

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

Arguments

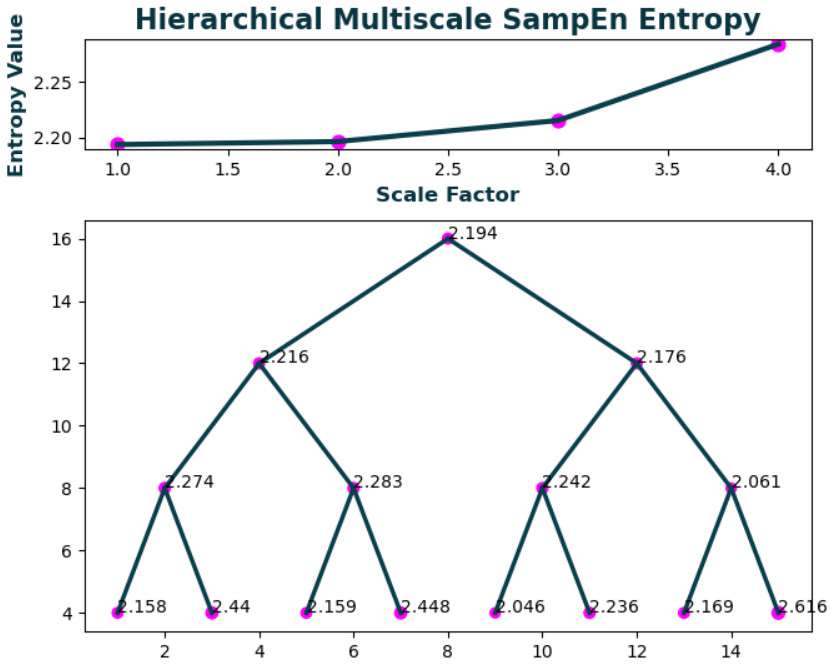
`Sig` Time series signal, a vector of length > 10.
The length of **`Sig`** (K) is halved at each scale. Only use the first 2^N data points are used such that $\min_N(K - 2^N)$.
i.e. For a signal of 5000 points, only the first 4096 points are used. For a signal of 1500 points, only the first 1024 points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

`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 [57]

3.4 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 two positional arguments:

the time series signals **`Sig`** (an Nx2 matrix),
and the multiscale entropy object, **`Mobj`**.

Examples (shown in Julia syntax):

Original multiscale cross-entropy [42]

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

Multiscale cross-distribution entropy using Rice's binning method and signal graining with empirical mode decomposition [47] [16]

```
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 [53] [15]

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

Refined-Composite multiscale cross-entropy calculated in dits [54]

```
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)
```

3.4.1 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.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.

3.4.2 XMSEn: Multiscale Cross-Entropy

Syntax

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

Arguments

Sig Time series signals, a N x 2 matrix where N > 10.

Scales Number of grained time scales, a positive integer.

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

"coarse" [42]

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

"modified"

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

"timeshift" [51]

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

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

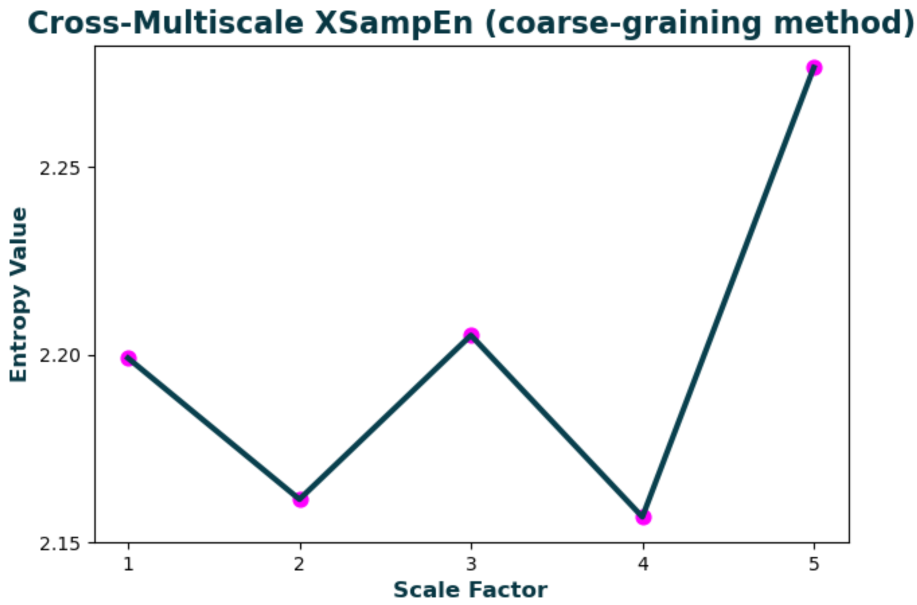
"imf" Grained time series at scale τ is the cumulative sum of intrinsic mode functions (IMF^1 to IMF^τ), where IMF^1 is the first sifting. [47]
***Note:** The empirical mode decomposition method use to derive the IMFs differs slightly between **MatLab, Python** and **Julia**, so **MSx** values will be 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 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 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 [42] [43] [44] [58] [59] [60] [61]

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

Syntax

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

Arguments

Sig Time series signals, a N x 2 matrix where N > 10.

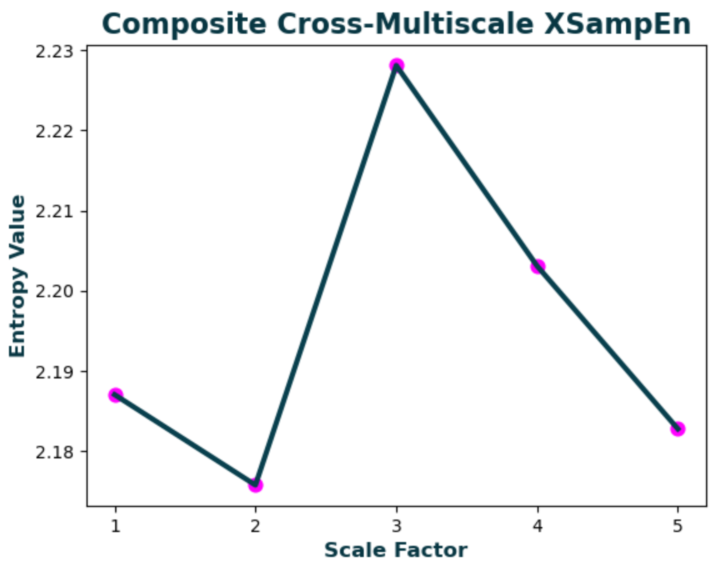
Scales Number of time scales, a positive integer.

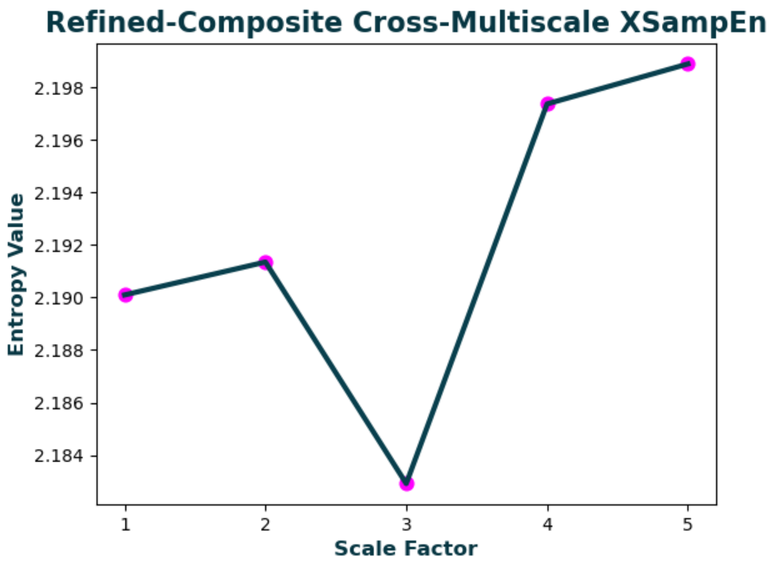
RadNew Radius rescaling method, an integer in the range [0 4].
When the **Cross**-entropy method specified by **Mobj** is **XSampEn** or **XApEn**, **RadNew** allows the radius threshold to be updated based on the grained signal at each time scale (X_τ). If a radius threshold value **(r)** is specified 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** 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 **XSampEn**, cXMSEn returns the refined-composite multiscale cross-entropy (**rcXMSEn**). [54]

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.





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 [58] [59] [60] [61] [53]

3.4.4 rXMSEn: Refined Multiscale Cross-Entropy

Syntax

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

Arguments

Sig Time series signals, a N x 2 matrix where N > 10.

Scales Number of time scales, a positive integer.

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

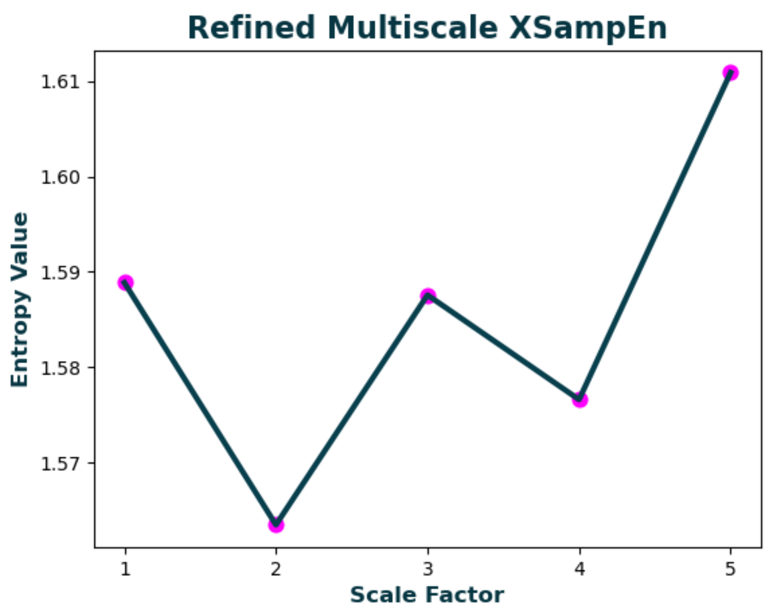
F_Num Numerator of Butterworth low-pass filter cutoff frequency, where $[0 < F_{Num} < 1]$. The cutoff frequency at each scale (τ) becomes: $F_c = \frac{F_{Num}}{\tau}$ (default: 0.5)

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 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 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 [42] [58] [55]

3.4.5 hXMSEn: Hierarchical Multiscale Cross-Entropy

Syntax

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

Arguments

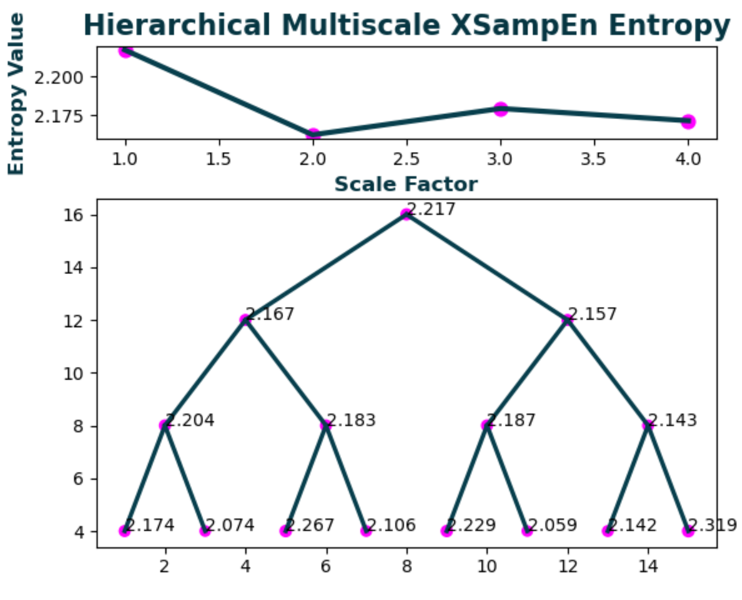
Sig Time series signals, a N x 2 matrix where N > 10.
The length of **Sig** (K) is halved at each scale. Only use the first 2^N data points are used such that $\min_N(K - 2^N)$.
i.e. For signals of 5000 points, only the first 4096 points are used. For signals of 1500 points, only the first 1024 points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

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 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 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 [57]

3.5 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`**) 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.***

3.5.1 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 [62]

3.5.2 FuzzEn2D: Bidimensional Fuzzy Entropy

Syntax

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

Arguments

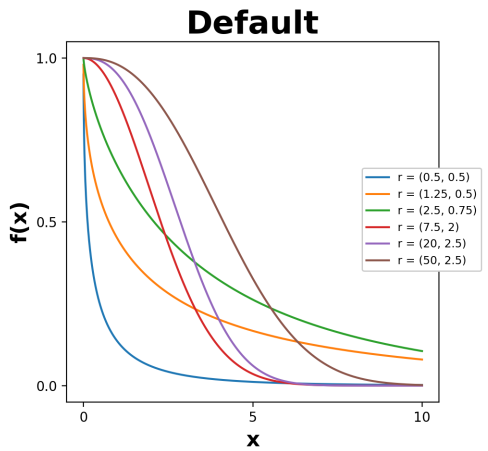
Mat N x M matrix, where N, M > 10.

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.

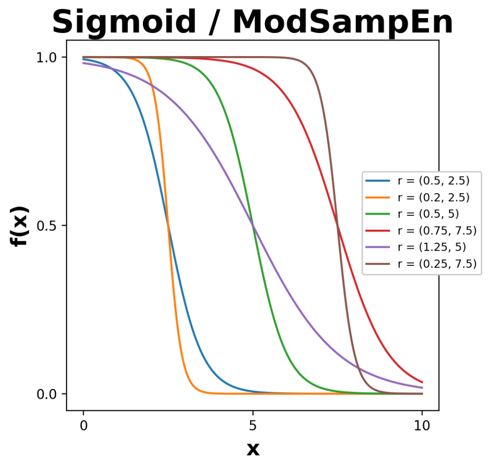
tau Time delay, a positive integer.

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

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

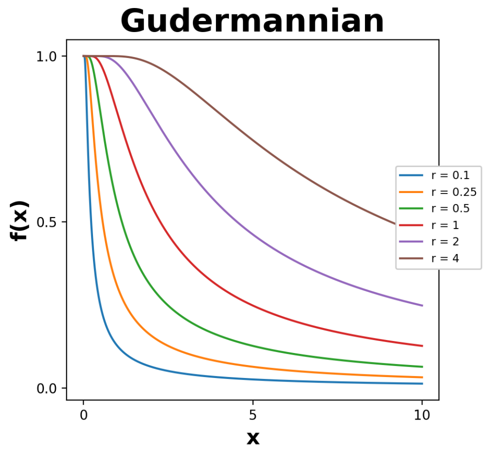


"sigmoid"/"modsampen" $f(x) = (1 + exp(\frac{x-r_2}{r_1}))^{-1}$



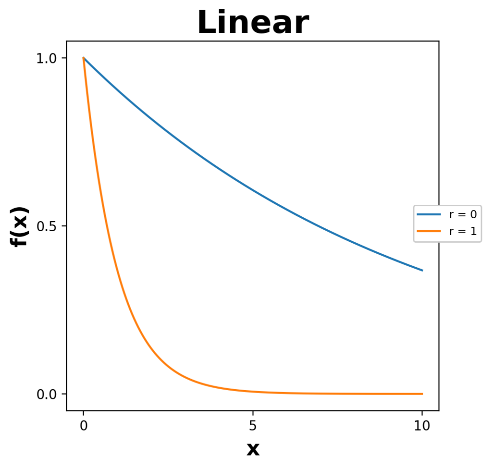
"gudermannian" $g(x) = atan(\frac{tanh(r_1)}{x})$
$f(x) = \frac{g(x)}{g(x_{max})}$

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



"linear"

If **r** = 0:
$f(x) = exp(-\frac{x - x_{min}}{x_{max} - x_{min}})$
If **r** = 1:
$f(x) = exp(-(x - x_{min}))$



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)
Gudermannian	A scalar value
Linear	0 or 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

Fuzz2D Bidimensional fuzzy entropy estimate.

References

[63], [64]

3.5.3 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 [65],

3.5.4 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.

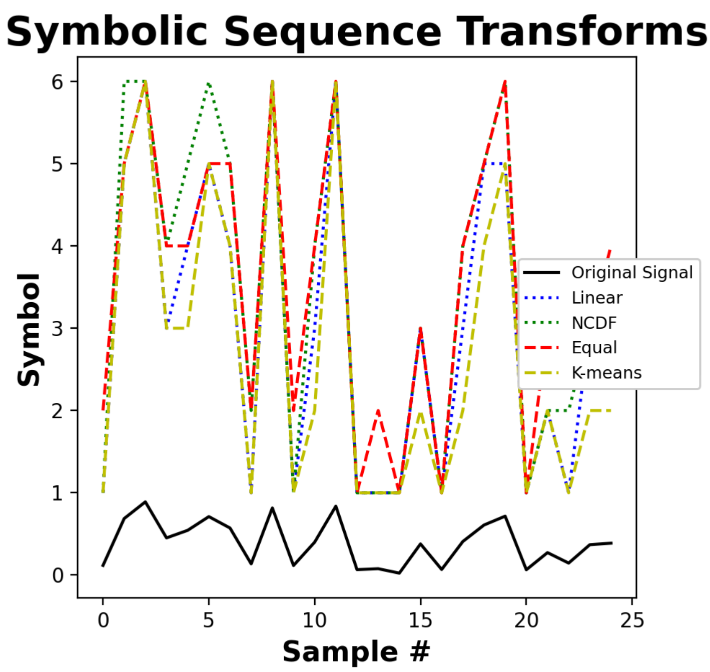
m Embedding dimension, [default: (floor(N/10) floor(M/10))]
- an integer > 1 for square submatrix embedding, or
- a two-element tuple of integers > 1 representing the height and width of the template submatrix.

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 [19]
"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 [22]
"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 [66],

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()` function from EntropyHub. To use this function as outlined in the examples below, **an internet connection is required.**
`ExampleData()` accepts any 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

THINGS TO REMEMBER

For **cross-entropy** and **multiscale cross-entropy** functions, the two time series signals are passed as a two-column or two-row matrix. At present, it is not possible to pass signals of different lengths separately.

Parameters of the base or cross- entropy methods are passed to multiscale and multiscale cross- functions using the multiscale entropy object using **`MSobject`**. 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 example in the following sections for more info.

Each bidimensional entropy function (**`SampEn2D, FuzzEn2D, DistEn2D`**) 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
```

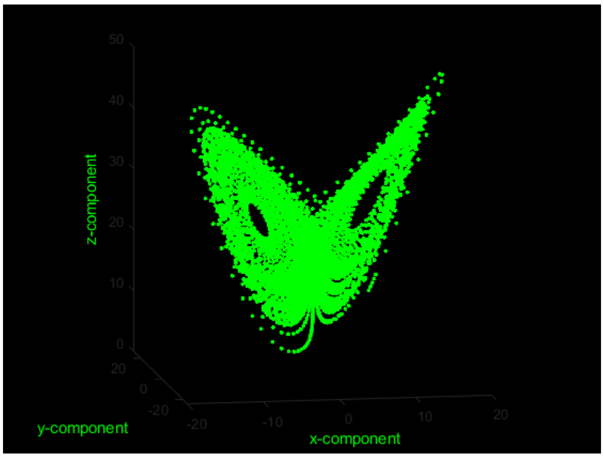
Calculate the sample entropy for each embedding dimension (m) from 0 to 4 with a time delay (tau) of 2 samples.

```
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
```



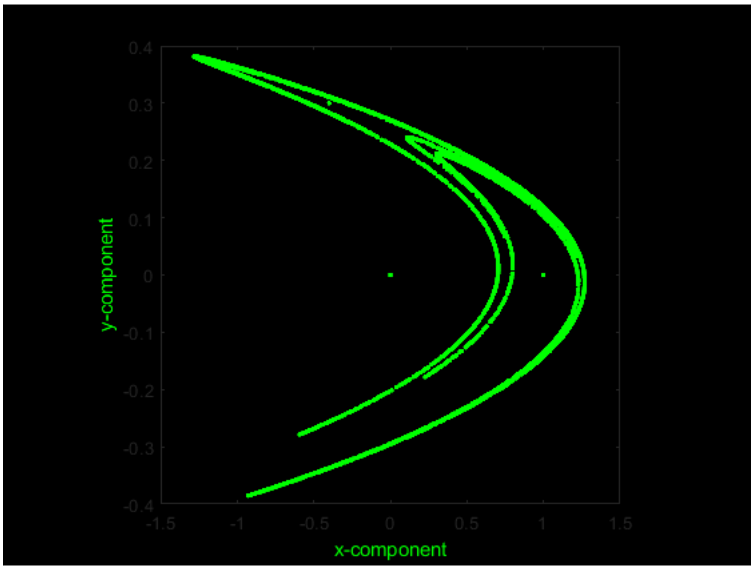
Calculate fine-grained permutation entropy of the z component in dits (logarithm base 10) with an embedding dimension of 3, time delay of 2, an alpha parameter of 1.234. Return `Pnorm` normalised w.r.t the number of all possible permutations (m!) and the condition permutation entropy (`cPE`) estimate.

```
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

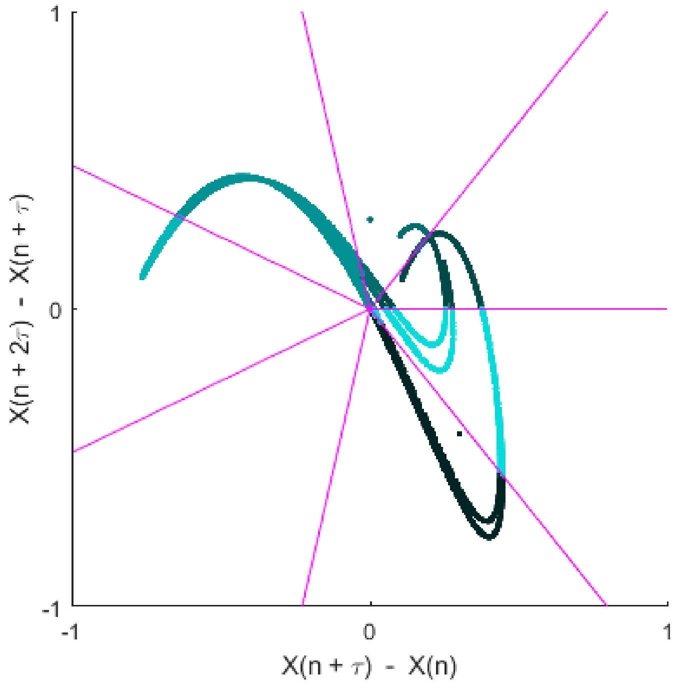
Import the x and y components of the Henon system of equations.

```
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
```



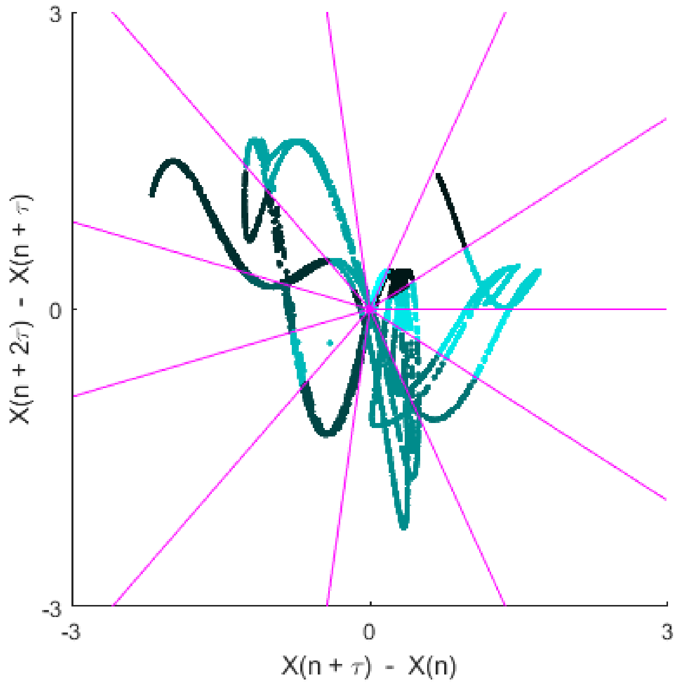
Calculate the phase entropy of the y-component in bits (logarithm base 2) without normalization using 7 angular partitions and return the second-order difference plot.

```
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, '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, '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
```

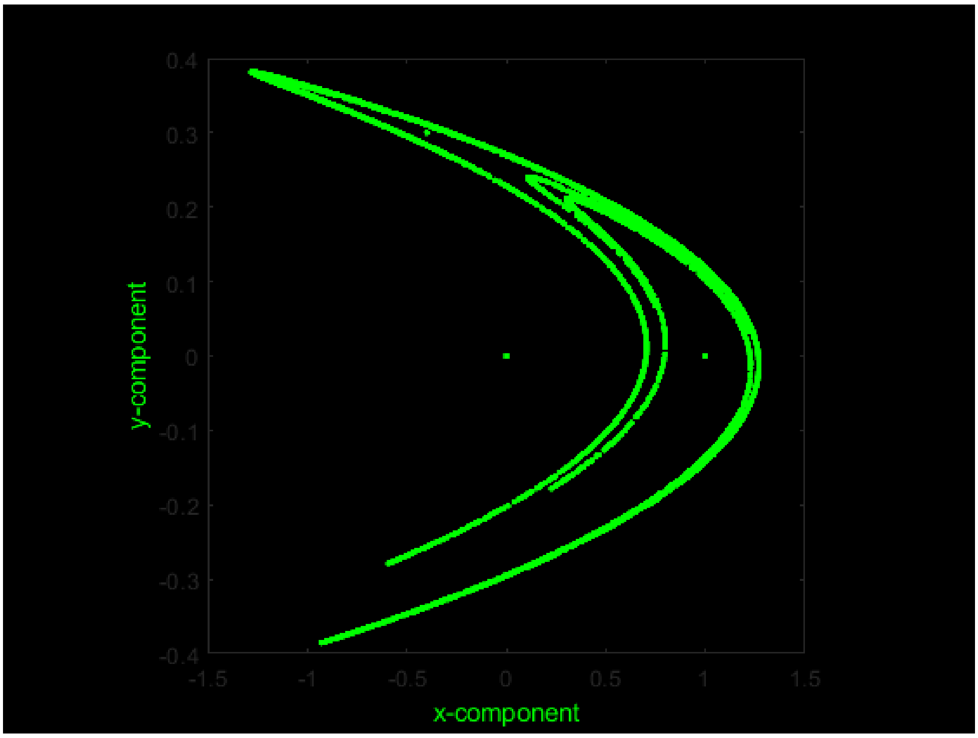
Calculate the comsposite multiscale cross-approximate entropy over 3 temporal scales where the radius distance threshold value (r) specified by Mobj becomes scaled by the variance of the signal at each scale.

```
MSx = cXMSEn(X, 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

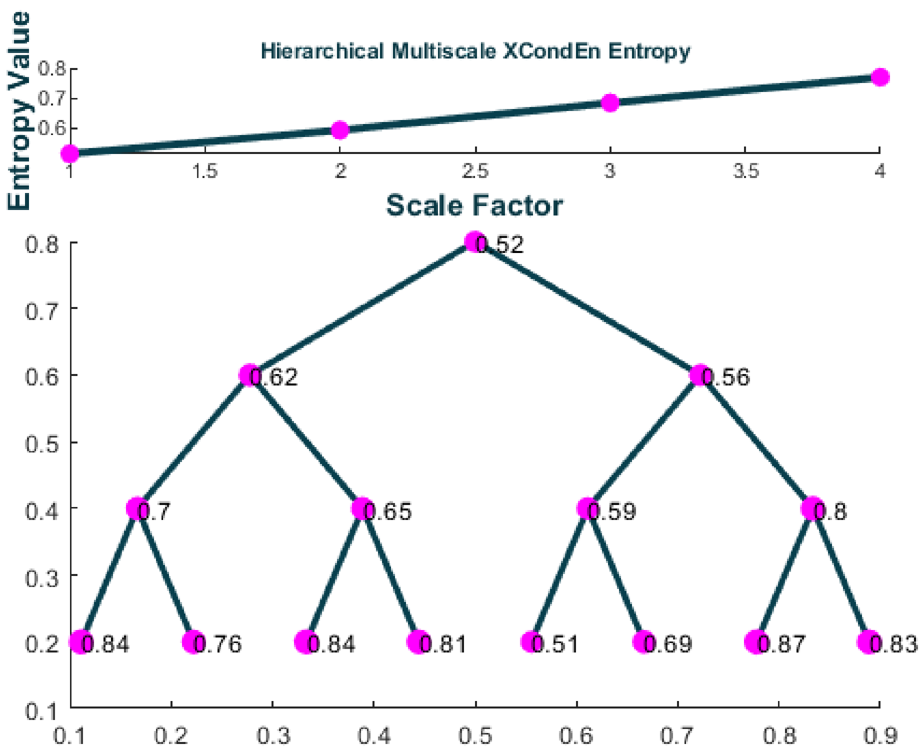
Import the x and y components of the Henon system of equations and create a multiscale entropy object with the following parameters:
EnType = XCondEn(), embedding dimension = 2, time delay = 2, number of symbols = 12, logarithm base = 2, normalization = true

```
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.

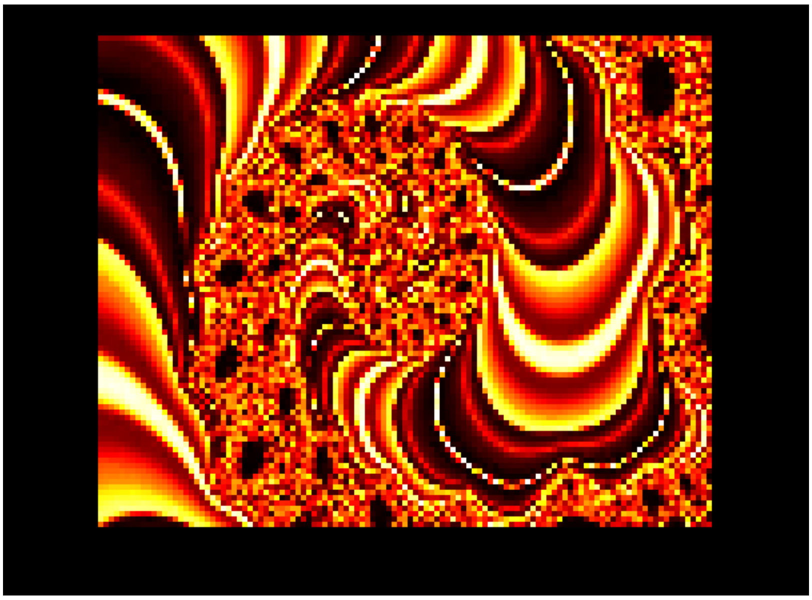
```
[MSx, Sn, Ci] = hXMSEn(Data, Mobj, 'Scales', 4, ...
   'Plotx', true)

>>> 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 'linear' fuzzy function with distances linearly normalised to the range [0, 1].
$f(x) = exp(-\frac{x-x_{min}}{x_{max}-x_{min}})$

```
FE2D = FuzzEn2D(X, 'm', [8 5], 'tau', 2, 'Fx', . . .
        'linear', 'r', 0, 'Logx', 3)

>>> FE2D =
        0.0016
```

4.2 Python:

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

```python
import EntropyHub
```

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

```
import EntropyHub as eh


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

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
```

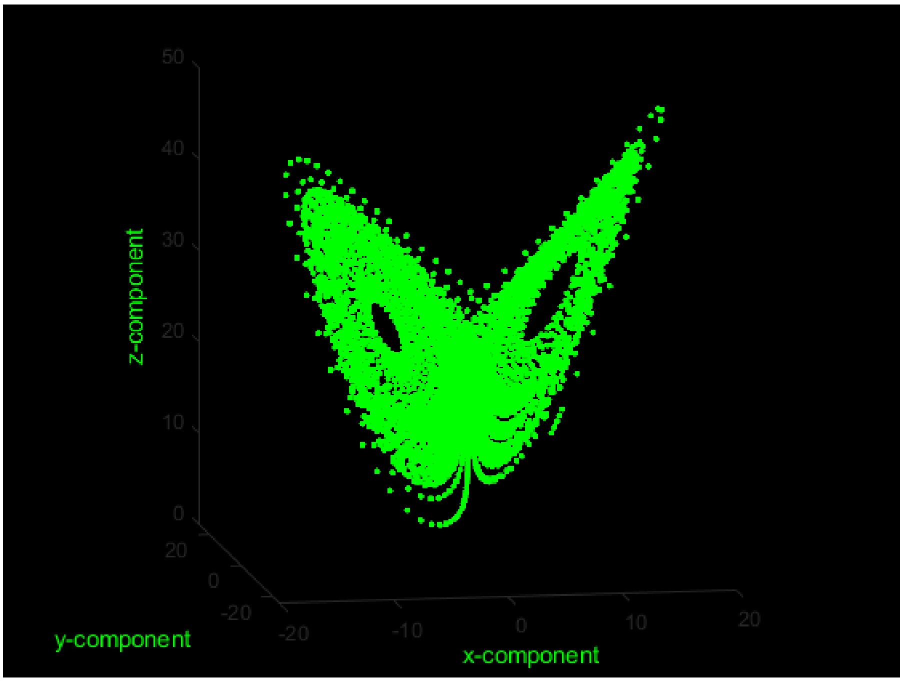
Calculate the sample entropy for each embedding dimension (m) from 0 to 4 with a time delay (tau) of 2 samples.

```
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.

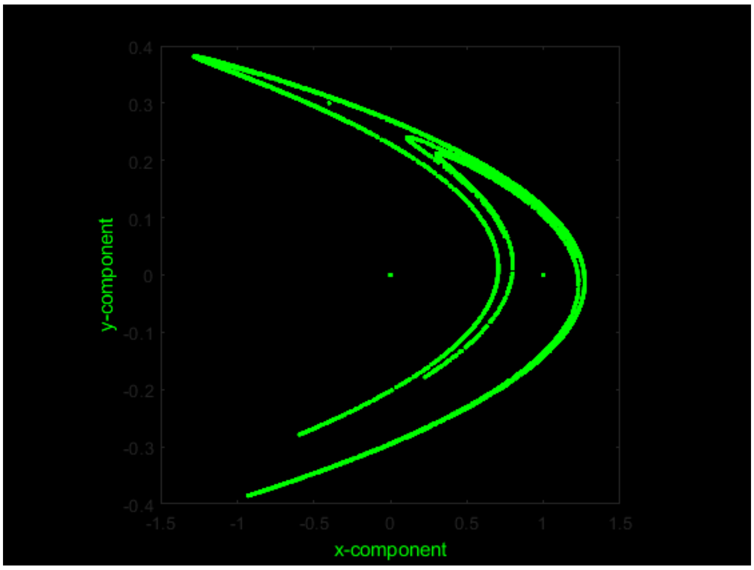
```
Z = Data[:,2];

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

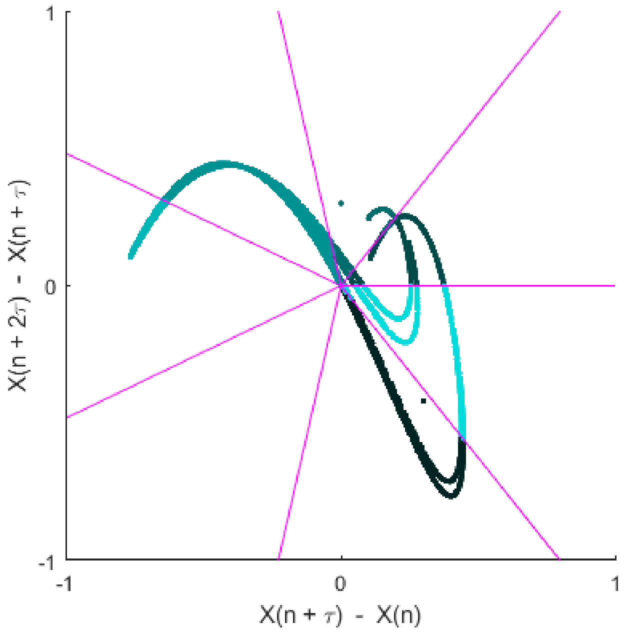
Import the x and y components of the Henon system of equations.

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

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



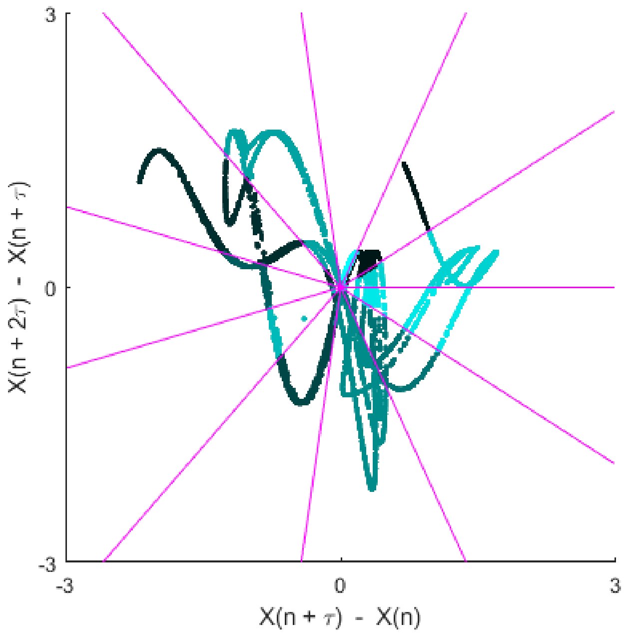
Calculate the phase entropy of the y-component in bits (logarithm base 2) without normalization using 7 angular partitions and return the second-order difference plot.

```
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, 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, 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}
```

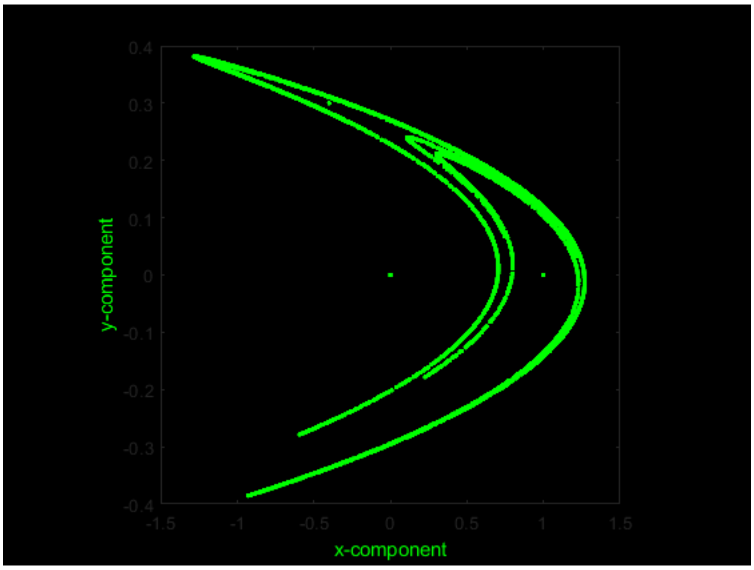
Calculate the comsposite multiscale cross-approximate entropy over 3 temporal scales where the radius distance threshold value (r) specified by Mobj becomes scaled by the variance of the signal at each scale.

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

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

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

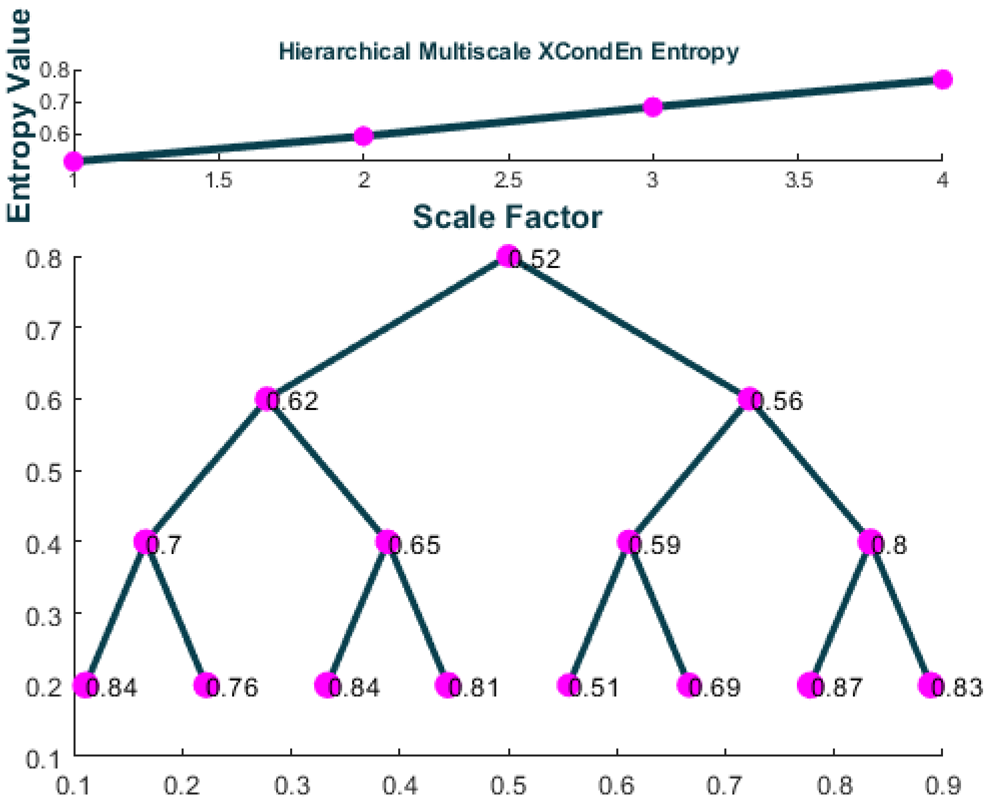
Import the x and y components of the Henon system of equations and create a multiscale entropy object with the following parameters:
EnType = XCondEn(), embedding dimension = 2, time delay = 2, number of symbols = 12, logarithm base = 2, normalization = true

```
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.

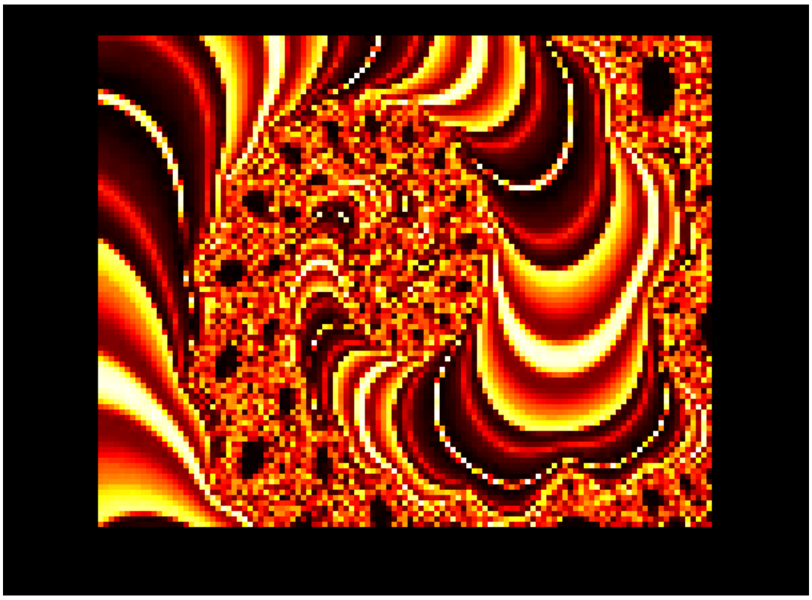
```
MSx, Sn, Ci = eh.hXMSEn(Data, Mobj, Scales = 4, Plotx = True)

>>> 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 'linear' fuzzy function with distances linearly normalised to the range [0, 1].
$f(x) = exp(-\frac{x-x_{min}}{x_{max}-x_{min}})$

```
FE2D = eh.FuzzEn2D(X, m = (8, 5), tau = 2, Fx = 'linear',
                                r = 0, Logx = 3)

>>> FE2D =
        0.00159093
```

4.3 Julia:

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

```julia
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
```

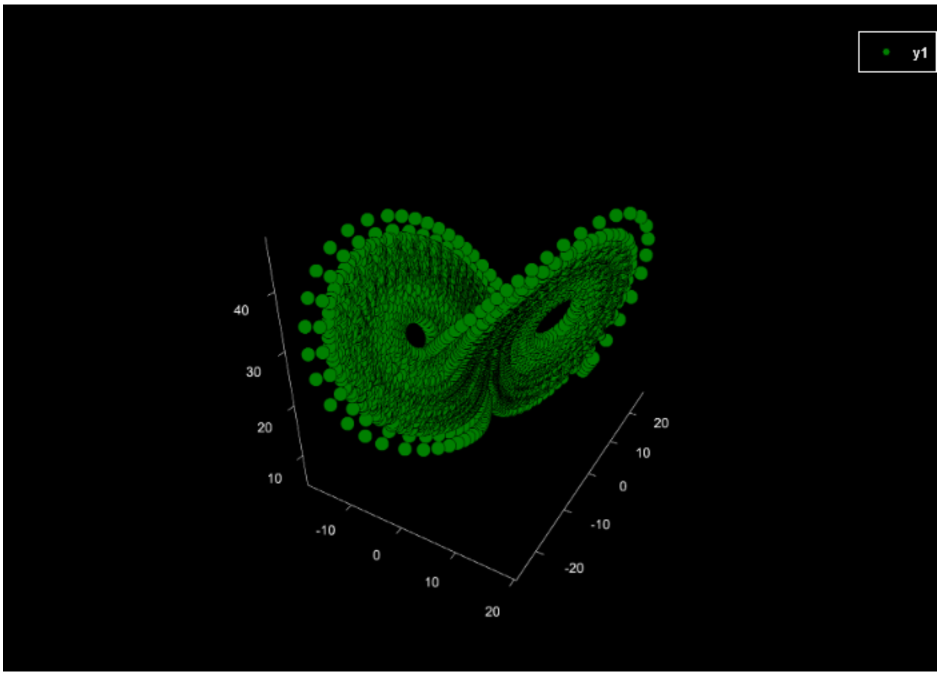
Calculate the sample entropy for each embedding dimension (m) from 0 to 4 with a time delay (tau) of 2 samples.

```
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)
```



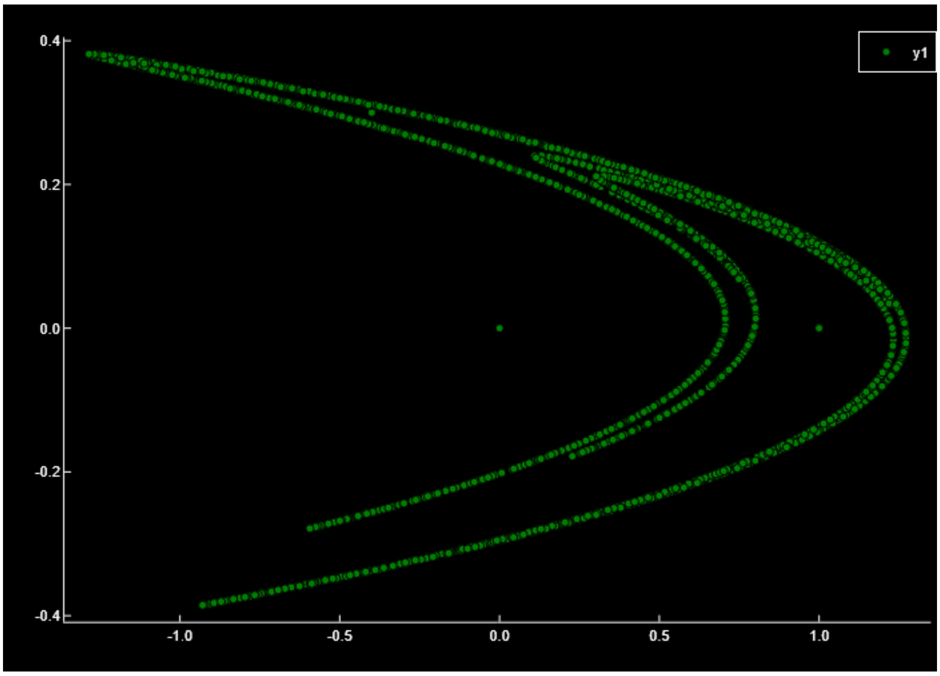
Calculate fine-grained permutation entropy of the z component in dits (logarithm base 10) with an embedding dimension of 3, time delay of 2, an alpha parameter of 1.234. Return `Pnorm` normalised w.r.t the number of all possible permutations (m!) and the condition permutation entropy (`cPE`) estimate.

```
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, 0.0781290449149276])
```

4.3.3 Example 3: Phase Entropy w/ Poincaré plot

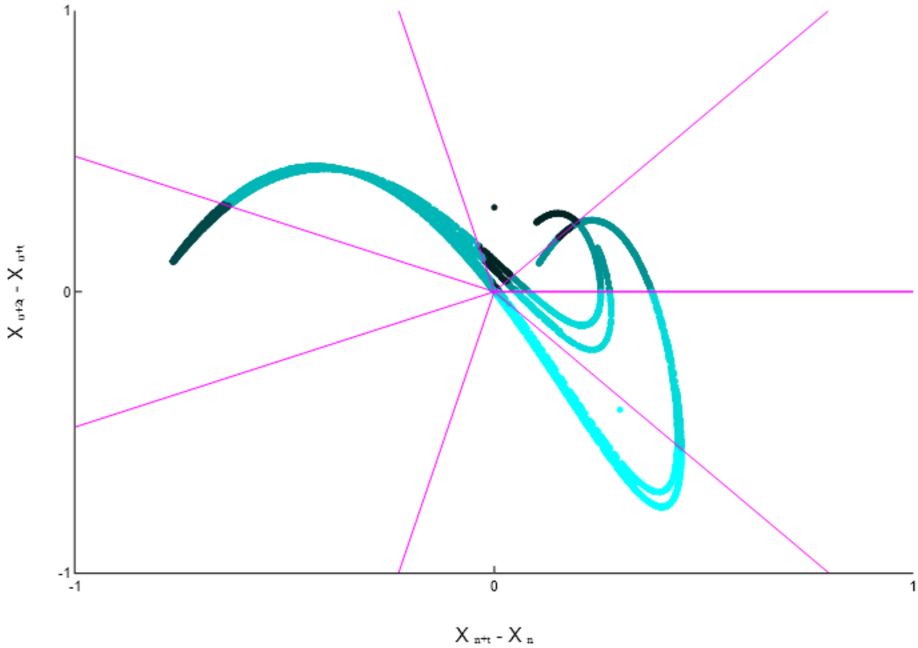
Import the x and y components of the Henon system of equations.

```
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)
```



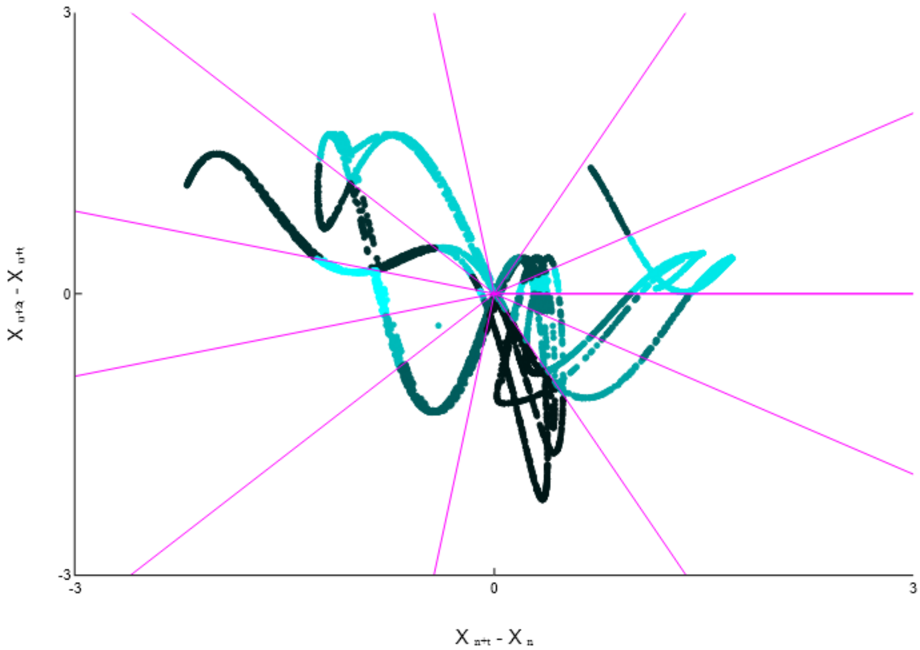
Calculate the phase entropy of the y-component in bits (logarithm base 2) without normalization using 7 angular partitions and return the second-order difference plot.

```
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, 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, 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)
```

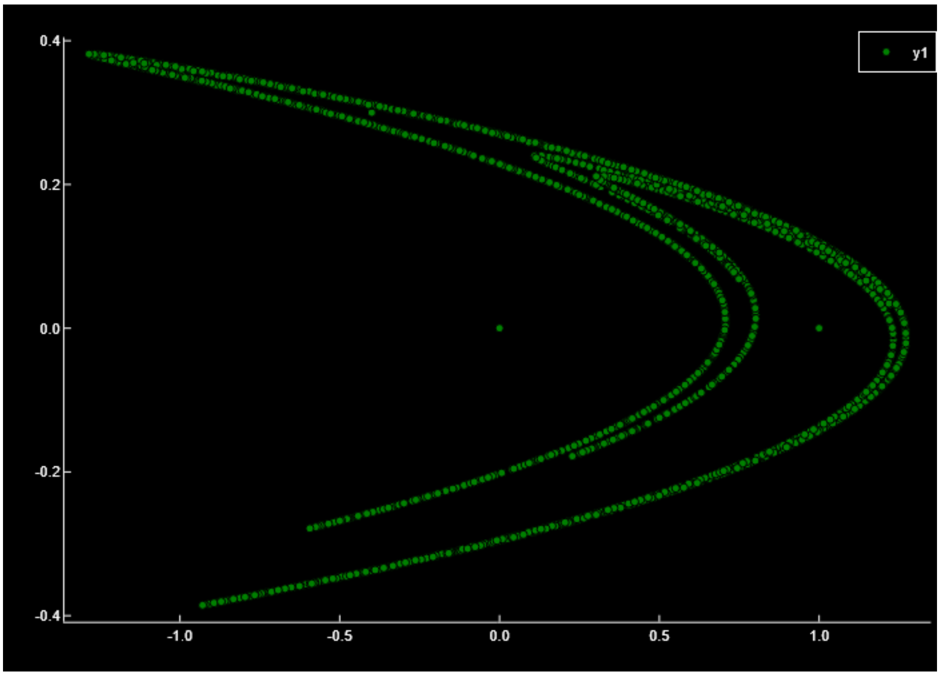
Calculate the comsposite multiscale cross-approximate entropy over 3 temporal scales where the radius distance threshold value (r) specified by Mobj becomes scaled by the variance of the signal at each scale.

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

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

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

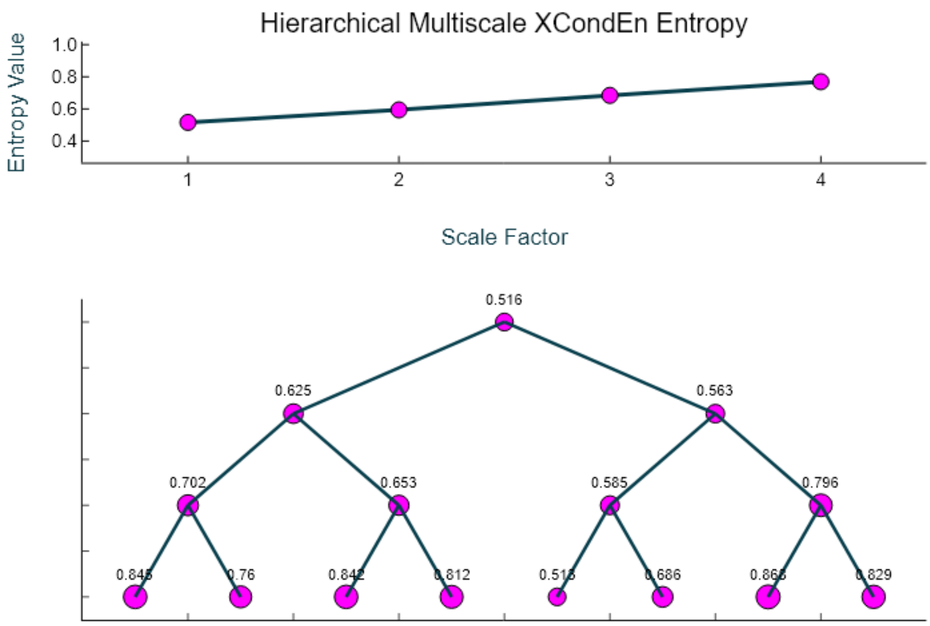
Import the x and y components of the Henon system of equations and create a multiscale entropy object with the following parameters:
EnType = XCondEn(), embedding dimension = 2, time delay = 2, number of symbols = 12, logarithm base = 2, normalization = true

```
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.

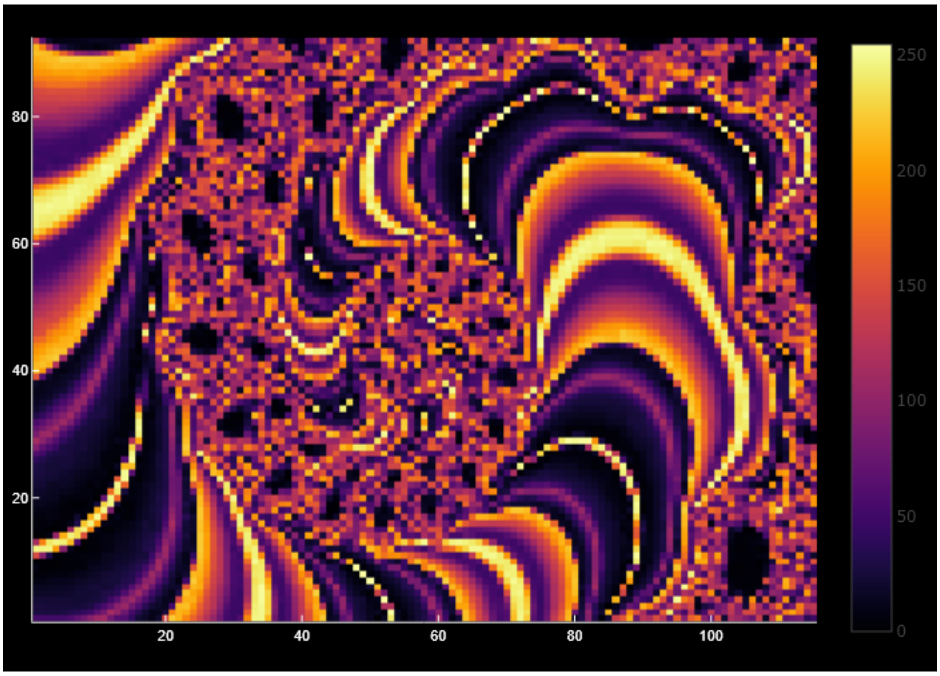
```
MSx, Sn, Ci = hXMSEn(Data, Mobj, Scales = 4, Plotx = true)
[Warning: Only first 4096 samples were used in hierarchical
|         decomposition.
|    The last 404 samples of the data sequence were ignored.
[@ EntropyHub._hXMSEn
. . . . . . . . . . . . . . . . . . . . . . .
([0.5159119, 0.62451155, 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 'linear' fuzzy function with distances linearly normalised to the range [0, 1].
$f(x) = exp(-\frac{x-x_{min}}{x_{max}-x_{min}})$

```
julia> FE2D = FuzzEn2D(X, m = (8, 5), tau = 2,
                       Fx = "linear", r = 0, Logx = 3)
0.0015909286623630356
```

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] 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.

[4] 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.

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

[6] 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

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

[8] 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.

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

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

[11] 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.

[12] 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

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

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

[15] Alberto Porta, et al.,
Measuring regularity by means of a corrected conditional entropy in sympathetic outflow,
Biological cybernetics 78.1 (1998): 71-78.

[16] 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.

[17] 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.

[18] 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.

[19] Mostafa Rostaghi and Hamed Azami,
Dispersion entropy: A measure for time-series analysis
IEEE Signal Processing Letters 23.5 (2016): 610-614.

[20] Hamed Azami and Javier Escudero,
Amplitude-and fluctuation-based dispersion entropy,
Entropy 20.3 (2018): 210.

[21] Li Yuxing, Xiang Gao and Long Wang,
Reverse dispersion entropy: A new complexity measure for sensor signal,
Sensors 19.23 (2019): 5203.

[22] 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.

[23] 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.

[24] 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.

[25] Venkatesh Rajagopalan and Asok Ray,
Symbolic time series analysis via wavelet-based partitioning,
Signal processing 86.11 (2006): 3309-3320

[26] Xiaofeng Liu, et al.,
Increment entropy as a measure of complexity for time series,
Entropy 18.1 (2016): 22.1.

[27] *** 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.

[28] Xiaofeng Liu, et al.,
Appropriate use of the increment entropy for electrophysiological time series,
Computers in biology and medicine 95 (2018): 13-23.

[29] Theerasak Chanwimalueang and Danilo Mandic,
Cosine similarity entropy: Self-correlation-based complexity analysis of dynamical systems,
Entropy 19.12 (2017): 652.

[30] Ashish Rohila and Ambalika Sharma,
Phase entropy: a new complexity measure for heart rate variability,
Physiological measurement 40.10 (2019): 105006.

[31] David Cuesta-Frau,
Slope Entropy: A New Time Series Complexity Estimator Based on Both Symbolic Patterns and Amplitude Information,
Entropy 21.12 (2019): 1167.

[32] 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.

[33] 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.

[34] Chang Yan, et al.,
Area asymmetry of heart rate variability signal,
Biomedical engineering online 16.1 (2017): 1-14.

[35] 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.

[36] C.K. Karmakar, A.H. Khandoker and M. Palaniswami,
Phase asymmetry of heart rate variability signal,
Physiological measurement 36.2 (2015): 303.

[37] Przemyslaw Guzik, et al.,
Heart rate asymmetry by Poincaré plots of RR intervals,
Biomedizinische Technik. Biomedical engineering 51.4 (2006): 272-275.

[38] Chang Francis Hsu, et al.,
Entropy of entropy: Measurement of dynamical complexity for biological systems,
Entropy 19.10 (2017): 550.

[39] Jiawei Yang, et al.,
Classification of Interbeat Interval Time-series Using Attention Entropy,
IEEE Transactions on Affective Computing (2020)

[40] 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.

[41] 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.

[42] Madalena Costa, Ary Goldberger, and C-K. Peng,
Multiscale entropy analysis of complex physiologic time series,
Physical review letters 89.6 (2002): 068102.

[43] Vadim V. Nikulin, and Tom Brismar,
Comment on "Multiscale entropy analysis of complex physiologic time series",
Physical review letters 92.8 (2004): 089803.

[44] Madalena Costa, Ary L. Goldberger, and C-K. Peng.
Costa, Goldberger, and Peng reply,
Physical Review Letters 92.8 (2004): 089804.

[45] Madalena Costa, Ary L. Goldberger and C-K. Peng,
Multiscale entropy analysis of biological signals,
Physical Review E 71.2 (2005): 021906

[46] 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.

[47] 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.

[48] Anne Humeau-Heurtier
The multiscale entropy algorithm and its variants: A review,
Entropy 17.5 (2015): 3110-3123.

[49] Jianbo Gao, et al.,
Multiscale entropy analysis of biological signals: a fundamental bi-scaling law,
Frontiers in computational neuroscience 9 (2015): 64.

[50] 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.

[51] Tuan D Pham,
Time-shift multiscale entropy analysis of physiological signals,
Entropy 19.6 (2017): 257.

[52] Hamed Azami and Javier Escudero,
Coarse-graining approaches in univariate multiscale sample and dispersion entropy,
Entropy 20.2 (2018): 138.

[53] Shuen-De Wu, et al.,
Time series analysis using composite multiscale entropy,
Entropy 15.3 (2013): 1069-1084.

[54] Shuen-De Wu, et al.,
Analysis of complex time series using refined composite multiscale entropy,
Physics Letters A 378.20 (2014): 1369-1374.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] Antoine Jamin and Anne Humeau-Heurtier,
(Multiscale) Cross-Entropy Methods: A Review,
Entropy 22.1 (2020): 45.

[60] 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.

[61] Yi Yin, Pengjian Shang, and Guochen Feng,
Modified multiscale cross-sample entropy for complex time series,
Applied Mathematics and Computation 289 (2016): 98-110.

[62] Luiz Eduardo Virgili Silva, et al.,
Two-dimensional sample entropy: Assessing image texture through irregularity, Biomedical Physics & Engineering Express 2.4 (2016): 045002.

[63] 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.

[64] Mirvana Hilal and Anne Humeau-Heurtier,
Bidimensional fuzzy entropy: Principle analysis and biomedical applications,
41st Annual International Conference of the IEEE (EMBC) Society 2019.

[65] 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.

[66] 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.

[67] Matthew W. Flood,
EntropyHub: An Open-Source Toolkit for Entropic Time Series Aalysis,
(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 / XCondEn / DispEn2D]

EnType — Type of *Base* or *Cross*- 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 / 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 / XApEn / XSampEn / XFuzzEn / XCondEn / XK2En / XPermEn / XDistEn] Number of grid partitions - [GridEn] Size of sub-matrix for bidimensional entropy estimation - [SampEn2D / FuzzEn2D / DistEn2D / DispEn2D]
Mat	Image (matrix) for estimating bidimensional entropies - [SampEn2D / DistEn2D / DispEn2D / FuzzEn2D]
Methodx	Time series graining method for multiscale entropy analysis - [MSEn / XMSEn]
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]
Plotx	Option to return a plot with the entropy value (method dependent) - [PhasEn / GridEn / MSEn / XMSEn / cMSEn / rMSEn / cXMSEn / rXMSEn / hMSEn / hXMSEn]
r	Distance threshold for matching vectors - [ApEn / SampEn / XApEn / XSampEn / SampEn2D] Angular threshold for matching vectors - [CoSiEn] Parameters for the fuzzy membership function - [FuzzEn / FuzzEn2D / XFuzzEn]
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

- [rMSEn / rXMSEn]

`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]

Type of symbolic sequence transform method
- [DispEn / SyDyEn / DispEn2D]

`tpx` Tuning parameter for the permutation entropy method specified by **`Typex`**
- [PermEn]