**(1)Sample entropy algorithms:**

**(2)Code from MATLAB Community:**

<https://www.mathworks.com/matlabcentral/fileexchange/35784-sample-entropy>

<https://www.mathworks.com/matlabcentral/fileexchange/69381-sample-entropy>

function value = sampen(signal, m, r, dist\_type)

% Error detection and defaults

if nargin < 3, error('Not enough parameters.'); end

if nargin < 4

dist\_type = 'chebychev';

fprintf('[WARNING] Using default distance method: chebychev.\n');

end

if ~isvector(signal)

error('The signal parameter must be a vector.');

end

if ~ischar(dist\_type)

error('Distance must be a string.');

end

if m > length(signal)

error('Embedding dimension must be smaller than the signal length (m<N).');

end

% Useful parameters

signal = signal(:)';

N = length(signal); % Signal length

sigma = std(signal); % Standard deviation

% Create the matrix of matches

matches = NaN(m+1,N);

for i = 1:1:m+1

matches(i,1:N+1-i) = signal(i:end);

end

matches = matches';

% Check the matches for m

d\_m = pdist(matches(:,1:m), dist\_type);

if isempty(d\_m)

% If B = 0, SampEn is not defined: no regularity detected

% Note: Upper bound is returned

value = Inf;

else

% Check the matches for m+1

d\_m1 = pdist(matches(:,1:m+1), dist\_type);

% Compute A and B

% Note: logical operations over NaN values are always 0

B = sum(d\_m <= r\*sigma);

A = sum(d\_m1 <= r\*sigma);

% Sample entropy value

% Note: norm. comes from [nchoosek(N-m+1,2)/nchoosek(N-m,2)]

value = -log((A/B)\*((N-m+1)/(N-m-1)));

end

% If A=0 or B=0, SampEn would return an infinite value. However, the

% lowest non-zero conditional probability that SampEn should

% report is A/B = 2/[(N-m-1)(N-m)]

if isinf(value)

% Note: SampEn has the following limits:

% - Lower bound: 0

% - Upper bound: log(N-m)+log(N-m-1)-log(2)

value = -log(2/((N-m-1)\*(N-m)));

end

end

**(3)Python code from wiki:**

**import** **numpy** **as** **np**

2

3

4 **def** sampen(L, m, r):

5 N = len(L)

6 B = 0.0

7 A = 0.0

8

9 *# Split time series and save all templates of length m*

10 xmi = np.array([L[i : i + m] **for** i **in** range(N - m)])

11 xmj = np.array([L[i : i + m] **for** i **in** range(N - m + 1)])

12

13 *# Save all matches minus the self-match, compute B*

14 B = np.sum([np.sum(np.abs(xmii - xmj).max(axis=1) <= r) - 1 **for** xmii **in** xmi])

15

16 *# Similar for computing A*

17 m += 1

18 xm = np.array([L[i : i + m] **for** i **in** range(N - m + 1)])

19

20 A = np.sum([np.sum(np.abs(xmi - xm).max(axis=1) <= r) - 1 **for** xmi **in** xm])

21

22 *# Return SampEn*

23 **return** -np.log(A / B)