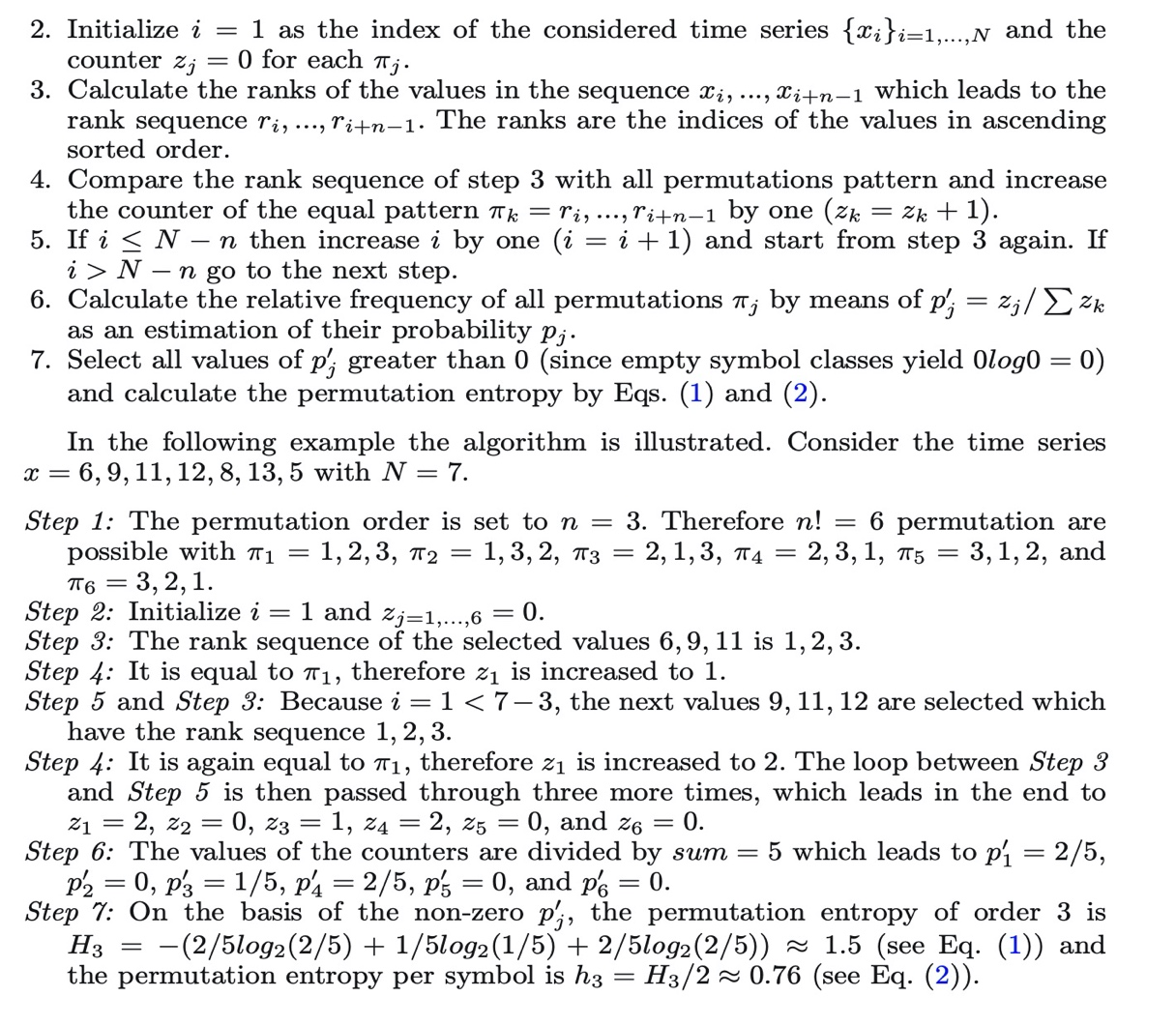
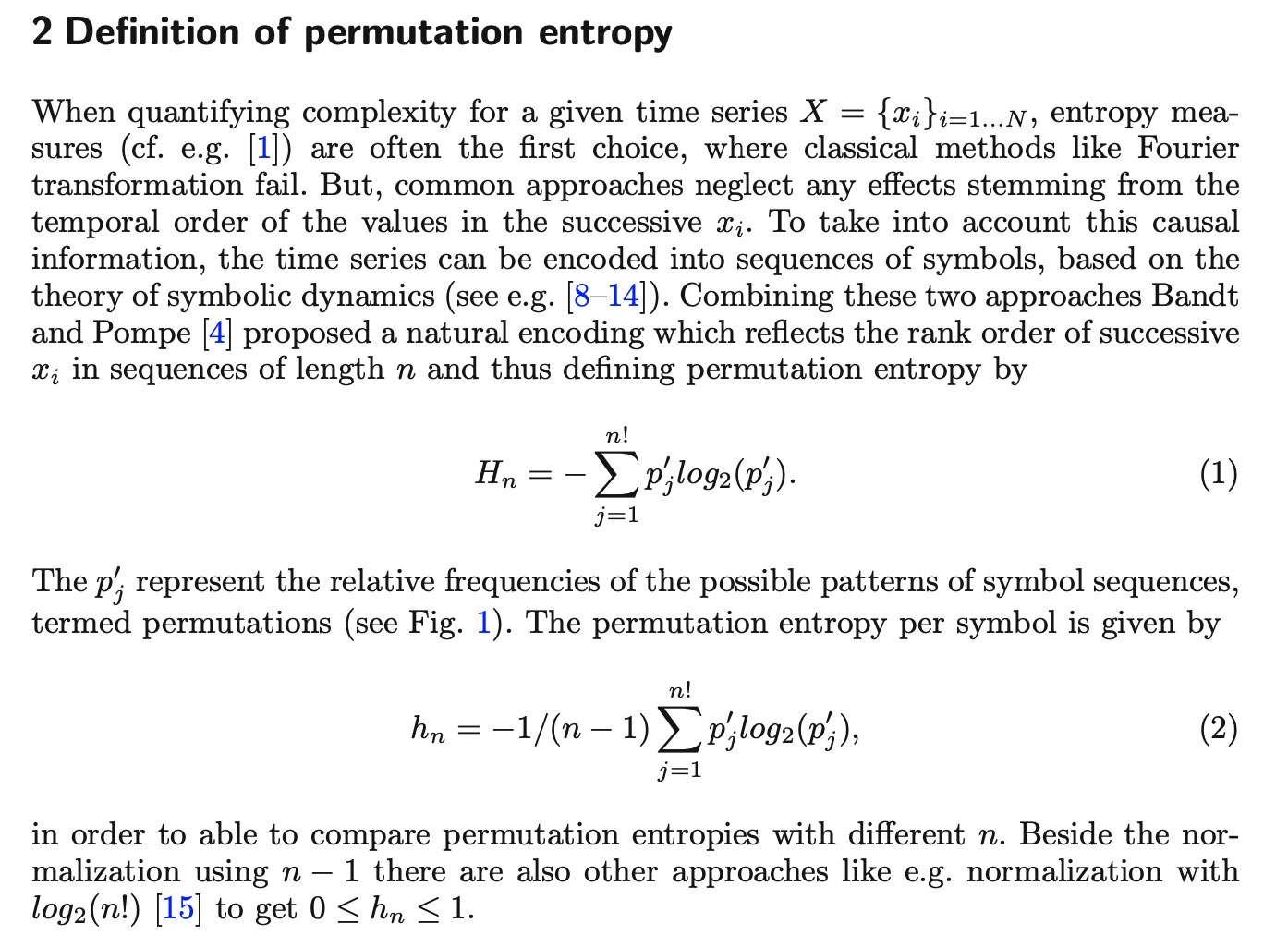
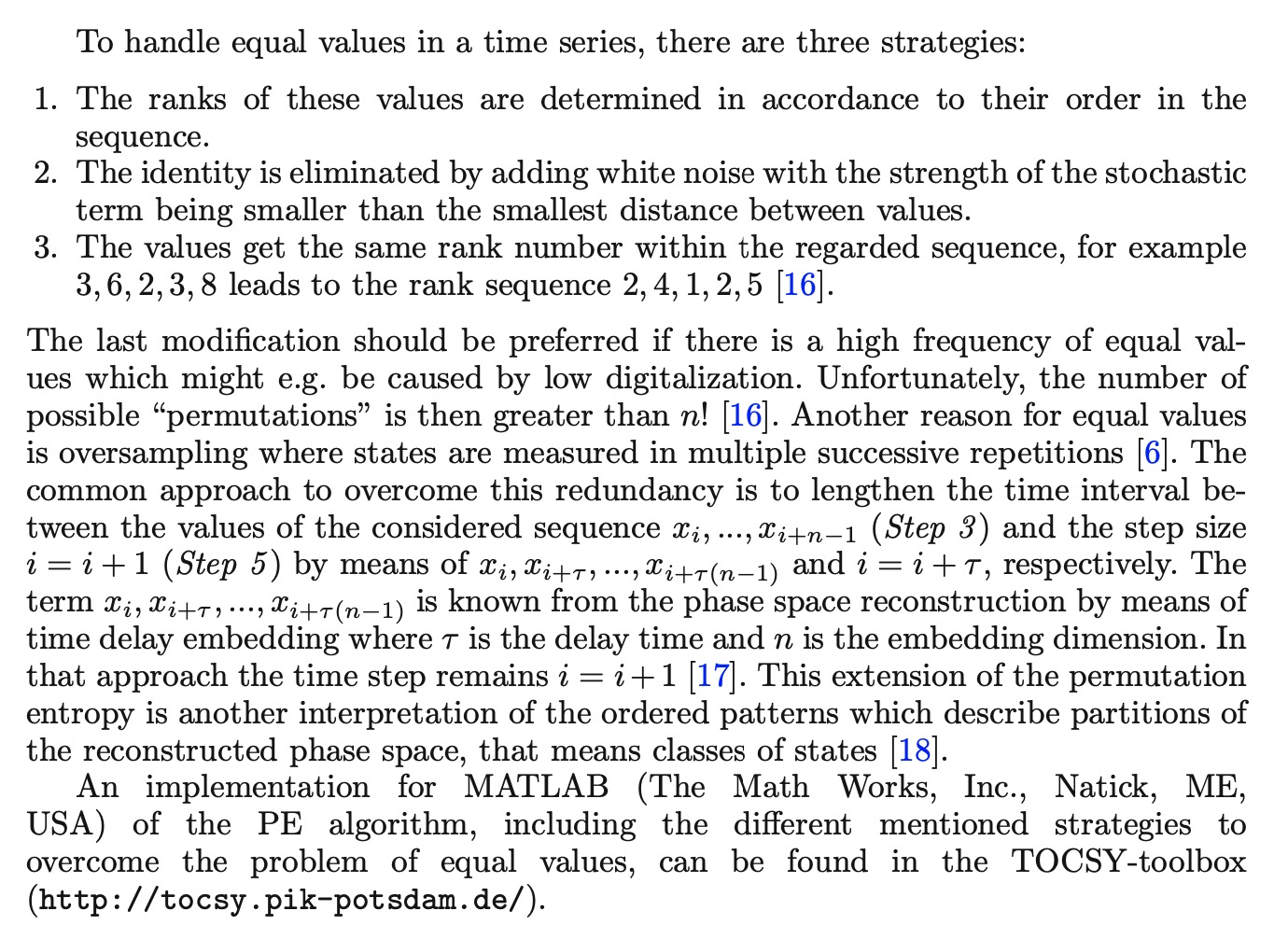
**(1)Permutation Entropy Algorithms:**

<https://www.aptech.com/blog/permutation-entropy/>

https://www.quora.com/What-is-permutation-entropy

(2)Code from MATLAB Community:

<https://www.mathworks.com/matlabcentral/fileexchange/44161-permutation-entropy-fast-algorithm>

function outdata = PE( indata, delay, order, windowSize )

% @brief PE efficiently [1] computes values of permutation entropy [2] in

% maximally overlapping sliding windows

%

% INPUT

% - indata - considered time series

% - delay - delay between points in ordinal patterns (delay = 1 means successive points)

% - order - order of the ordinal patterns (order+1 - number of points in ordinal patterns)

% - windowSize - size of sliding window

% OUTPUT

% - outdata - values of normalised permutation entropy as defined in [2]

%

% REFERENCES

% [1] Unakafova, V.A., Keller, K., 2013. Efficiently measuring complexity

% on the basis of real-world data. Entropy, 15(10), 4392-4415.

% [2] Bandt C., Pompe B., 2002. Permutation entropy: a natural complexity

% measure for time series. Physical review letters, APS

%

% @author Valentina Unakafova

% @date 10.11.2017

% @email UnakafovaValentina(at)gmail.com

load( ['table' num2str( order ) '.mat'] ); % the precomputed table

patternsTable = eval( ['table' num2str( order )] );

nPoints = numel( indata ); % length of the time series

opOrder1 = order + 1;

orderDelay = order\*delay;

nPatterns = factorial( opOrder1 ); % amount of ordinal patterns

patternsDistribution = zeros( 1, nPatterns ); % distribution of ordinal patterns

outdata = zeros( 1, nPoints ); % values of permutation entropy

inversionNumbers = zeros( 1, order ); % inversion numbers of ordinal pattern (i1,i2,...,id)

prevOP = zeros( 1, delay ); % previous ordinal patterns for 1:opDelay

ordinalPatternsInWindow = zeros( 1, windowSize ); % ordinal patterns in the window

ancNum = nPatterns./factorial( 2:opOrder1 ); % ancillary numbers

peTable( 1:windowSize ) = -( 1:windowSize ).\*log( 1:windowSize ); % table of precomputed values

peTable( 2:windowSize ) = ( peTable( 2:windowSize ) - peTable( 1:windowSize - 1 ) )./windowSize;

for iTau = 1:delay

cnt = iTau;

inversionNumbers( 1 ) = ( indata( orderDelay + iTau - delay ) >= indata( orderDelay + iTau ) );

for j = 2:order

inversionNumbers( j ) = sum( indata( ( order - j )\*delay + iTau ) >= ...

indata( ( opOrder1 - j )\*delay + iTau:delay:orderDelay + iTau ) );

end

ordinalPatternsInWindow( cnt ) = sum( inversionNumbers.\*ancNum ); % first ordinal patterns

patternsDistribution( ordinalPatternsInWindow( cnt )+ 1 ) = ...

patternsDistribution( ordinalPatternsInWindow( cnt ) + 1 ) + 1;

for j = orderDelay + delay + iTau:delay:windowSize + orderDelay % loop for the first window

cnt = cnt + delay;

posL = 1; % the position of the next point

for i = j - orderDelay:delay:j - delay

if( indata( i ) >= indata( j ) )

posL = posL + 1;

end

end

ordinalPatternsInWindow( cnt ) = ...

patternsTable( ordinalPatternsInWindow( cnt - delay )\*opOrder1 + posL );

patternsDistribution( ordinalPatternsInWindow( cnt ) + 1 ) = ...

patternsDistribution( ordinalPatternsInWindow( cnt ) + 1 ) + 1;

end

prevOP( iTau ) = ordinalPatternsInWindow( cnt );

end

ordDistNorm = patternsDistribution/windowSize;

tempSum = 0;

for iPattern = 1:nPatterns

if ( ordDistNorm( iPattern ) ~= 0 )

tempSum = tempSum - ordDistNorm( iPattern )\*log( ordDistNorm( iPattern ) );

end

end

outdata( windowSize + delay\*order ) = tempSum;

iTau = mod( windowSize, delay ) + 1; % current shift 1:delay

patternPosition = 1; % position of the current pattern in the window

for t = windowSize + delay\*order + 1:nPoints % loop over all points

posL = 1; % the position of the next point

for j = t-orderDelay:delay:t-delay

if( indata( j ) >= indata( t ) )

posL = posL + 1;

end

end

nNew = patternsTable( prevOP( iTau )\*opOrder1 + posL ); % incoming ordinal pattern

nOut = ordinalPatternsInWindow( patternPosition ); % outcoming ordinal pattern

prevOP( iTau ) = nNew;

ordinalPatternsInWindow( patternPosition ) = nNew;

nNew = nNew + 1;

nOut = nOut + 1;

if ( nNew ~= nOut ) % update the distribution of ordinal patterns

patternsDistribution( nNew ) = patternsDistribution( nNew ) + 1; % incoming ordinal pattern

patternsDistribution( nOut ) = patternsDistribution( nOut ) - 1; % outcoming ordinal pattern

outdata( t ) = outdata( t - 1 ) + ( peTable( patternsDistribution( nNew ) ) - ...

peTable( patternsDistribution( nOut ) + 1 ) );

else

outdata( t ) = outdata( t - 1 );

end

iTau = iTau + 1;

patternPosition = patternPosition + 1;

if ( iTau > delay )

iTau = 1;

end

if ( patternPosition > windowSize )

patternPosition = 1;

end

end

outdata = outdata( windowSize + delay\*order:end )/log( factorial( order + 1 ) );

**(3)Code from github by Python:**

<https://github.com/srk-srinivasan/Permutation-Entropy>