OctaveMatlabExamples

Joseph Lizier edited this page on 22 Jul · 5 revisions

Examples of using the toolkit in Octave or Matlab

Demos > Octave/Matlab code examples

Octave/Matlab code examples

This page describes a basic set of demonstration scripts for using the toolkit in Octave or Matlab. The `.m` files can be found at demos/octave in the svn or main distributions. Please note that other more complicated examples are available from the main Demos page. These examples have been confirmed to work in both Octave and Matlab.

Please see UseInOctaveMatlab for instructions on how to begin using the java toolkit from inside octave or matlab.

This page contains the following code examples:

- Example 1 - Transfer entropy on binary data
- Example 2 - Transfer entropy on multidimensional binary data
- Example 3 - Transfer entropy on continuous data using kernel estimators
- Example 4 - Transfer entropy on continuous data using Kraskov estimators
- Example 5 - Multivariate transfer entropy on binary data
- Example 6 - Dynamic dispatch with Mutual info calculator
- Example 7 - coming soon
- Example 8 - coming soon
- Example 9 - Transfer entropy on multivariate continuous data using Kraskov estimators
- Example 10 - Conditional Transfer entropy on continuous multivariate data using Kraskov estimators

Be aware

1. In octave conversion between native octave array types and java arrays is not straightforward (particularly for multidimensional arrays, or int arrays). Octave often reports that a method is

not found. One could directly convert each element in an octave array to a java array first; however we recommend using the supplied scripts described in OctaveJavaArrayConversion (and see example use in Example 2 and Example 5). These scripts are called here so that the code is runnable in Octave or Matlab - inside Matlab they simply return the array that was passed as input.

2. In java arrays are indexed from 0, whereas in octave or Matlab you are used to indexing them from 1. So when you call a method such as MatrixUtils `.select(double[]() data, int fromIndex, int length)` , you must be aware that fromIndex will be indexed from 0 inside the toolkit, not 1!!

Example 1 - Transfer entropy on binary data

example1TeBinaryData.m - Simple transfer entropy (TE) calculation on binary data using the discrete TE calculator:

```
% Change location of jar to match yours:
javaaddpath('../../infodynamics.jar');

% Generate some random binary data.
% Note that we need the *1 to make this a number not a Boolean,
%  otherwise this will not work (as it cannot match the method signature)
sourceArray=(rand(100,1)>0.5)*1;
destArray = [0; sourceArray(1:99)];
sourceArray2=(rand(100,1)>0.5)*1;
% Create a TE calculator and run it:
teCalc=javaObject('infodynamics.measures.discrete.TransferEntropyCalculatorDiscrete'
teCalc.initialise();
% Since we have simple arrays of doubles, we can directly pass these in:
teCalc.addObservations(sourceArray, destArray);
fprintf('For copied source, result should be close to 1 bit : ');
result = teCalc.computeAverageLocalOfObservations()
teCalc.initialise();
teCalc.addObservations(sourceArray2, destArray);
fprintf('For random source, result should be close to 0 bits: ');
result2 = teCalc.computeAverageLocalOfObservations()
```

Example 2 - Transfer entropy on multidimensional binary data

example2TeMultidimBinaryData.m - Simple transfer entropy (TE) calculation on multidimensional binary data using the discrete TE calculator.

This example is important for Octave users, because it shows how to handle multidimensional arrays from Octave to Java (this is not as simple as single dimensional arrays in example 1 - it requires using supplied scripts to convert the array).

```
% Change location of jar to match yours:
javaaddpath('../../infodynamics.jar');

% Create many columns in a multidimensional array,
%  where the next time step (row 2) copies the value of the column on the left
%  from the previous time step (row 1):
twoDTimeSeriesOctave       = (rand(1, 100)>0.5)*1;
twoDTimeSeriesOctave(2, :) = [twoDTimeSeriesOctave(1,100), twoDTimeSeriesOctave(1, 1

% Things get a little tricky if we want to pass 2D arrays into Java.
% Unlike native Octave 1D arrays in Example 1,
%  native Octave 2D+ arrays do not seem to get directly converted to java arrays,
%  so we use the supplied scripts to make the conversion (via org.octave.Matrix clas
% Matlab handles the conversion automatically, so in Matlab this script just returns
%  the array that was passed in.
twoDTimeSeriesJavaInt = octaveToJavaIntMatrix(twoDTimeSeriesOctave);

% Create a TE calculator and run it:
teCalc=javaObject('infodynamics.measures.discrete.TransferEntropyCalculatorDiscrete'
teCalc.initialise();
% Add observations of transfer across one cell to the right per time step:
teCalc.addObservations(twoDTimeSeriesJavaInt, 1);
fprintf('The result should be close to 1 bit here, since we are executing copy opera
result2D = teCalc.computeAverageLocalOfObservations()
```

Example 3 - Transfer entropy on continuous data using kernel estimators

example3TeContinuousDataKernel.m - Simple transfer entropy (TE) calculation on continuous-valued data using the (box) kernel-estimator TE calculator.

```
% Change location of jar to match yours:
javaaddpath('../../infodynamics.jar');

% Generate some random normalised data.
numObservations = 1000;
covariance=0.4;
sourceArray=randn(numObservations, 1);
destArray = [0; covariance*sourceArray(1:numObservations-1) + (1-covariance)*randn(n
sourceArray2=randn(numObservations, 1); % Uncorrelated source
% Create a TE calculator and run it:
teCalc=javaObject('infodynamics.measures.continuous.kernel.TransferEntropyCalculator
teCalc.setProperty('NORMALISE', 'true'); % Normalise the individual variables
teCalc.initialise(1, 0.5); % Use history length 1 (Schreiber k=1), kernel width of 0
teCalc.setObservations(sourceArray, destArray);
% For copied source, should give something close to 1 bit:
result = teCalc.computeAverageLocalOfObservations();
fprintf('TE result %.4f bits; expected to be close to %.4f bits for these correlated
```

```
    result, log(1/(1-covariance^2))/log(2));
teCalc.initialise(); % Initialise leaving the parameters the same
teCalc.setObservations(sourceArray2, destArray);
% For random source, it should give something close to 0 bits
result2 = teCalc.computeAverageLocalOfObservations();
fprintf('TE result %.4f bits; expected to be close to 0 bits for uncorrelated Gaussi
    result2);

% We can get insight into the bias by examining the null distribution:
nullDist = teCalc.computeSignificance(100);
fprintf(['Null distribution for unrelated source and destination ', ...
    '(i.e. the bias) has mean %.4f and standard deviation %.4f\n'], ...
    nullDist.getMeanOfDistribution(), nullDist.getStdOfDistribution());
```

Example 4 - Transfer entropy on continuous data using Kraskov estimators

example4TeContinuousDataKraskov.m - Simple transfer entropy (TE) calculation on continuous-valued data using the Kraskov-estimator TE calculator.

```
% Change location of jar to match yours:
javaaddpath('../../infodynamics.jar');

% Generate some random normalised data.
numObservations = 1000;
covariance=0.4;
sourceArray=randn(numObservations, 1);
destArray = [0; covariance*sourceArray(1:numObservations-1) + (1-covariance)*randn(n
sourceArray2=randn(numObservations, 1); % Uncorrelated source
% Create a TE calculator and run it:
teCalc=javaObject('infodynamics.measures.continuous.kraskov.TransferEntropyCalculato
teCalc.initialise(1); % Use history length 1 (Schreiber k=1)
teCalc.setProperty('k', '4'); % Use Kraskov parameter K=4 for 4 nearest points
% Perform calculation with correlated source:
teCalc.setObservations(sourceArray, destArray);
result = teCalc.computeAverageLocalOfObservations();
% Note that the calculation is a random variable (because the generated
%  data is a set of random variables) - the result will be of the order
%  of what we expect, but not exactly equal to it; in fact, there will
%  be a large variance around it.
fprintf('TE result %.4f nats; expected to be close to %.4f nats for these correlated
    result, log(1/(1-covariance^2)));
% Perform calculation with uncorrelated source:
teCalc.initialise(); % Initialise leaving the parameters the same
teCalc.setObservations(sourceArray2, destArray);
result2 = teCalc.computeAverageLocalOfObservations();
fprintf('TE result %.4f nats; expected to be close to 0 nats for these uncorrelated

% We can also compute the local TE values for the time-series samples here:
```

```
%  (See more about utility of local TE in the CA demos)
localTE = teCalc.computeLocalOfPreviousObservations();
fprintf('Notice that the mean of locals, %.4f nats, equals the previous result\n', .
    sum(javaMatrixToOctave(localTE))/(numObservations-1));
```

Example 5 - Multivariate transfer entropy on binary data

example5TeBinaryMultivarTransfer.m - Multivariate transfer entropy (TE) calculation on binary data using the discrete TE calculator.

```
% Change location of jar to match yours:
javaaddpath('../../infodynamics.jar');

% Generate some random binary data.
% Note that we need the *1 to make this a number not a Boolean,
%  otherwise this will not work (as it cannot match the method signature)
numObservations = 100;
sourceArray=(rand(numObservations,2)>0.5)*1;
sourceArray2=(rand(numObservations,2)>0.5)*1;
% Destination variable takes a copy of the first bit of the source in bit 1,
%  and an XOR of the two bits of the source in bit 2:
destArray = [0, 0; sourceArray(1:numObservations-1, 1), xor(sourceArray(1:numObserva
% Create a TE calculator and run it:
teCalc=javaObject('infodynamics.measures.discrete.TransferEntropyCalculatorDiscrete'
teCalc.initialise();
% We need to construct the joint values of the dest and source before we pass them i
%  and need to use the matrix conversion routine when calling from Matlab/Octave:
mUtils= javaObject('infodynamics.utils.MatrixUtils');
teCalc.addObservations(mUtils.computeCombinedValues(octaveToJavaDoubleMatrix(sourceA
        mUtils.computeCombinedValues(octaveToJavaDoubleMatrix(destArray), 2));
fprintf('For source which the 2 bits are determined from, result should be close to
result = teCalc.computeAverageLocalOfObservations()
teCalc.initialise();
teCalc.addObservations(mUtils.computeCombinedValues(octaveToJavaDoubleMatrix(sourceA
        mUtils.computeCombinedValues(octaveToJavaDoubleMatrix(destArray), 2));
fprintf('For random source, result should be close to 0 bits in theory: ');
result2 = teCalc.computeAverageLocalOfObservations()
fprintf('\nThe result for random source is inflated towards 0.3 due to finite observ
```

Example 6 - Dynamic dispatch with Mutual info calculator

example6DynamicCallingMutualInfo.m - This example shows how to write Matlab/Octave code to take advantage of the common interfaces defined for various information-theoretic calculators. Here, we use the common form of the `infodynamics.measures.continuous.MutualInfoCalculatorMultiVariate` interface (which is never named here) to write common code for a mutual information calculation into which we can plug one of three concrete implementations (kernel estimator, Kraskov estimator or linear-Gaussian estimator) by dynamically supplying the class name of the concrete implementation.

```
% Change location of jar to match yours:
javaaddpath('../../infodynamics.jar');

%---------------------
% 1. Properties for the calculation (these are dynamically changeable, you could
%    load them in from another properties file):
% The name of the data file (relative to this directory)
datafile = '../data/4ColsPairedNoisyDependence-1.txt';
% List of column numbers for univariate time seres 1 and 2:
%  (you can select any columns you wish to be contained in each variable)
univariateSeries1Column = 1; % array indices start from 1 in octave/matlab
univariateSeries2Column = 3;
% List of column numbers for joint variables 1 and 2:
%  (you can select any columns you wish to be contained in each variable)
jointVariable1Columns = [1,2]; % array indices start from 1 in octave/matlab
jointVariable2Columns = [3,4];
% The name of the concrete implementation of the interface
%  infodynamics.measures.continuous.MutualInfoCalculatorMultiVariate
%  which we wish to use for the calculation.
% Note that one could use any of the following calculators (try them all!):
%  implementingClass = 'infodynamics.measures.continuous.kraskov.MutualInfoCalculato
%  implementingClass = 'infodynamics.measures.continuous.kernel.MutualInfoCalculator
%  implementingClass = 'infodynamics.measures.continuous.gaussian.MutualInfoCalculat
implementingClass = 'infodynamics.measures.continuous.kraskov.MutualInfoCalculatorMu

%---------------------
% 2. Load in the data
data = load(datafile);
% Pull out the columns from the data set for a univariate MI calculation:
univariateSeries1 = data(:, univariateSeries1Column);
univariateSeries2 = data(:, univariateSeries2Column);
% Pull out the columns from the data set for a multivariate MI calculation:
jointVariable1 = data(:, jointVariable1Columns);
jointVariable2 = data(:, jointVariable2Columns);

%---------------------
% 3. Dynamically instantiate an object of the given class:
% (in fact, all java object creation in octave/matlab is dynamic - it has to be,
%  since the languages are interpreted. This makes our life slightly easier at this
%  point than it is in demos/java/lateBindingDemo where we have to handle this manua
miCalc = javaObject(implementingClass);

%---------------------
% 4. Start using the MI calculator, paying attention to only
%  call common methods defined in the interface type
%  infodynamics.measures.continuous.MutualInfoCalculatorMultiVariate
%  not methods only defined in a given implementation class.
% a. Initialise the calculator for a univariate calculation:
miCalc.initialise(1, 1);
% b. Supply the observations to compute the PDFs from:
```

```
miCalc.setObservations(octaveToJavaDoubleArray(univariateSeries1), octaveToJavaDoubl
% c. Make the MI calculation:
miUnivariateValue = miCalc.computeAverageLocalOfObservations();

%---------------------
% 5. Continue onto a multivariate calculation, still
%    only calling common methods defined in the interface type.
% a. Initialise the calculator for a multivariate calculation
%   to use the required number of dimensions for each variable:
miCalc.initialise(length(jointVariable1Columns), length(jointVariable2Columns));
% b. Supply the observations to compute the PDFs from:
%    (Note the different method call octaveToJavaDoubleMatrix to convert a *multivar
%     time-series to Java format here)
miCalc.setObservations(octaveToJavaDoubleMatrix(jointVariable1), octaveToJavaDoubleM
% c. Make the MI calculation:
miJointValue = miCalc.computeAverageLocalOfObservations();

fprintf('MI calculator %s\ncomputed the univariate MI(%d;%d) as %.5f and joint MI([%
        implementingClass, univariateSeries1Column, univariateSeries2Column, miUniva
        sprintf('%d,', jointVariable1Columns), sprintf('%d,', jointVariable2Columns)
```

Example 9 - Transfer entropy on multivariate continuous data using Kraskov estimators

example9TeContinuousMultivariateDataKraskov.m - Transfer entropy (TE) calculation on *multivariate* continuous-valued data using the Kraskov-estimator TE calculator (i.e. computing transfer from a multivariate source to a multivariate destination). The original code for this example was contributed by Viola Priesemann.

```
% Change location of jar to match yours:
javaaddpath('../../infodynamics.jar')

% Generate some random normalised data.
numObservations = 10000;
covariance=0.4;

% Define the dimension of the states of the RVs
sourceDim = 2;
destDim = 3;

sourceMVArray = randn(numObservations, sourceDim);
% Set first two columns of dest to copy source values
destMVArray  = [zeros(1,sourceDim); covariance*(sourceMVArray(1:numObservations-1,:)
% Set a third colum to be randomised
destMVArray(:,3) = randn(numObservations, 1);
sourceMVArray2= randn(numObservations, sourceDim); % Uncorrelated source

% Create a TE calculator and run it:
teCalc=javaObject('infodynamics.measures.continuous.kraskov.TransferEntropyCalculato
```

```
teCalc.initialise(1,sourceDim,destDim); % Use history length 1 (Schreiber k=1)
teCalc.setProperty('k', '4'); % Use Kraskov parameter K=4 for 4 nearest points
teCalc.setObservations(octaveToJavaDoubleMatrix(sourceMVArray), octaveToJavaDoubleMa
% Perform calculation with correlated source:
result = teCalc.computeAverageLocalOfObservations();
% Note that the calculation is a random variable (because the generated
%  data is a set of random variables) - the result will be of the order
%  of what we expect, but not exactly equal to it; in fact, there will
%  be some variance around it. It will probably be biased down here
%  due to small correlations between the supposedly uncorrelated variables.
fprintf('TE result %.4f nats; expected to be close to %.4f nats for the two correlat
result, 2*log(1/(1-covariance^2)));

% Perform calculation with uncorrelated source:
teCalc.initialise(1,sourceDim,destDim); % Initialise leaving the parameters the same
teCalc.setObservations(octaveToJavaDoubleMatrix(sourceMVArray2), octaveToJavaDoubleM
result2 = teCalc.computeAverageLocalOfObservations();
fprintf('TE result %.4f nats; expected to be close to 0 nats for these uncorrelated
clear teCalc
```

Example 10 - Conditional Transfer entropy on continuous multivariate data using Kraskov estimators

Available from distributions 1.4 onwards.

example10ConditionalTeKraskov.m - Conditional Transfer entropy (TE) calculation on multivariate continuous-valued data using the Kraskov-estimator *Conditional* TE calculator (i.e. computing transfer from a multivariate source to a multivariate destination). The original code for this example is based on discussions with Roberto Sotero Diaz.

```
% Change location of jar to match yours:
javaaddpath('../../infodynamics.jar')

% Generate some random normalised data.
numObservations = 100000;
% Keep the sum of squares of these covariances below 1 to allow proper calculation o
covarianceToSource = 0.4;
covarianceToConds = [0.3,0.3];
if (sumsq([covarianceToSource, covarianceToConds]) >= 1)
    error('Sum of squares of the covariances must be < 1 here');
end
noiseCovar = sqrt(1 - sumsq([covarianceToSource, covarianceToConds]));

% Generate the random variables
sourceArray = randn(numObservations, 1);
condArray = randn(numObservations, length(covarianceToConds));
destArray = [0; covarianceToSource*sourceArray(1:numObservations-1,:) + covarianceTo

% Expected results:
```

```
expectedConditional = -0.5 * log(noiseCovar .* noiseCovar ./ (1 - sumsq(covarianceTo
expectedPairwise = -0.5 * log(1-covarianceToSource.^2);

% Create a conditional TE calculator and run it:
teCalc=javaObject('infodynamics.measures.continuous.kraskov.ConditionalTransferEntro
teCalc.initialise(1,1, ... % Destination embedding length (Schreiber k=1) and delays
        1,1, ...   % Source embedding length (Schreiber l=1) and delays
        1, ...     % Source-destination delay of 1 (default)
        octaveToJavaDoubleArray([1,1]), ... % Embedding lengths for each conditional
        octaveToJavaDoubleArray([1,1]), ... % Embedding delays for each conditional
        octaveToJavaDoubleArray([1,1]) ... % conditional-destination delays for each
);
teCalc.setObservations(octaveToJavaDoubleArray(sourceArray), ...
    octaveToJavaDoubleArray(destArray), ...
    octaveToJavaDoubleMatrix(condArray));
% Perform calculation with correlated source, but no conditioning on other sources:
conditionalResult = teCalc.computeAverageLocalOfObservations();

% Create a pairwise TE calculator and run it:
teCalc=javaObject('infodynamics.measures.continuous.kraskov.TransferEntropyCalculato
teCalc.initialise(); % Use default embeddings of 1 (e.g. Schreiber k=1 and l=1)
teCalc.setObservations(octaveToJavaDoubleArray(sourceArray), octaveToJavaDoubleArray
% Perform calculation with correlated source, but no conditioning on other sources:
pairwiseResult = teCalc.computeAverageLocalOfObservations();

% Note that the calculation is a random variable (because the generated
%  data is a set of random variables) - the result will be of the order
%  of what we expect, but not exactly equal to it; in fact, there will
%  be some variance around it. It will probably be biased down here
%  due to small correlations between the supposedly uncorrelated variables.
fprintf('From %d samples:\nTE result conditional result = %.4f nats, pairwise = %.4f
    numObservations, conditionalResult, pairwiseResult, expectedConditional, expecte
```

JIDT -- Java Information Dynamics Toolkit -- Joseph Lizier *et al.*