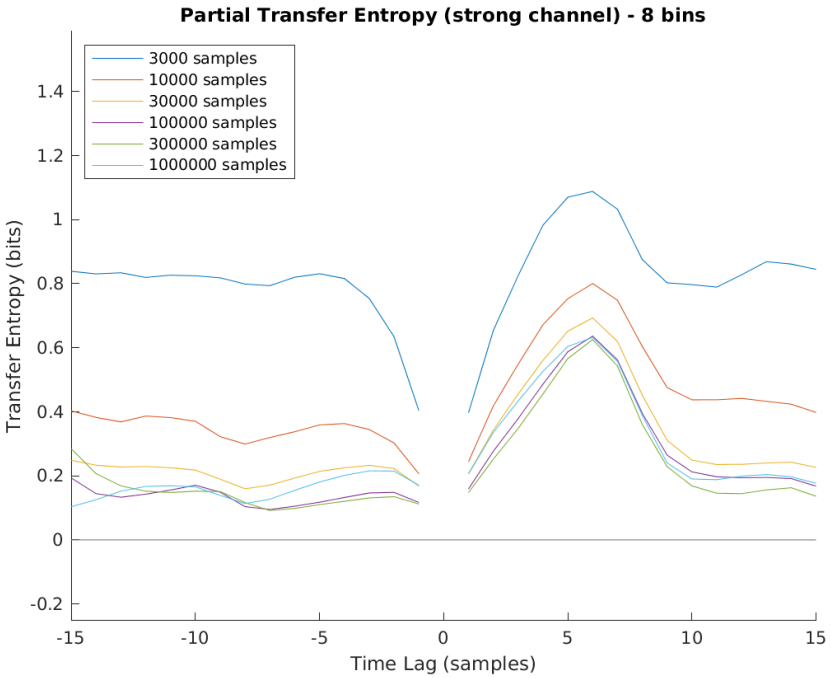
Chris's Entropy Library - Function Reference

Written by Christopher Thomas – April 23, 2024.



Contents

Chapter 1

Data Structures and Additional Notes

1.1 EXTRAPOLATION.txt

```
Extrapolation (via cEn_calcExtrapWrapper) attempts to curve fit the output
of a data processing function (typically an entropy calculation function)
at various small sample counts to estimate the output at large sample counts.


Extrapolation is performed using the method described in Palmigiano 2017,
which is based on the method described in Strong 1998, which is based on
the analysis performed in Treves 1995:

(Palmigiano 2017) "Flexible Information Routing by Transient Synchrony"
(Strong 1998) "Entropy and Information in Neural Spike Trains"
(Treves 1995) "The Upward Bias in Measures of Information Derived From
  Limited Data Samples"


For several values of N, M subsets of size (nsamples/N) are selected from
the sample data. The entropy calculation is run on these M subsets and the
results averaged, producing a result for the chosen value of N. A
A quadratic fit is performed to the results (as a function of N), and the
Y intercept (N = 0) is taken as the estimated infinite-length output.


An "extrapolation parameter structure" is a structure with the following
fields:

"divisors" is a vector containing divisors for nsamples.   Default: [1:10]
"testcount" is the number of tests to average for each divisor.   Default: 3

NOTE - If there are fewer than three divisors, the order of the curve fit
will be reduced. A noteworthy use of this is divisors = [ 1 ], testcount
= 1 to return the calculation result without extrapolation.
```

(This is the end of the file.)

Chapter 2

Function Reference

2.1 cEn_calcConditionalShannon.m

```
% function bits = cEn_calcConditionalShannon( dataseries, bins, exparams )
%
% This calculates the conditional entropy associated with a set of signals.
% This is the average amount of additional information that a sample from
% variable Y gives you when you already know variables X_1..X_k.
%
% This needs a large number of samples to generate accurate results. To
% compensate for smaller sample counts, this may optionally use the
% extrapolation method described in EXTRAPOLATION.txt.
%
% "dataseries" is a (Nchans,Nsamples) matrix containing several data series.
%   The first series (chan = 1) is the variable Y; remaining series are X_k.
% "bins" is a scalar or vector (to generate histogram bins) or a cell array
%   (to supply bin edge lists). If it's a vector of length Nchans or a
%   scalar, it indicates how many bins to use for each channel's data. If
%   it's a cell array, bins{chanidx} provides the list of edges used for
%   binning each channel's data.
% "exparams" is a an optional structure containing extrapolation tuning
%   parameters, per EXTRAPOLATION.txt. If this is empty, default parameters
%   are used. If this is absent, no extrapolation is performed.
%
% "bits" is a scalar with the average additional entropy provided by an
%   observation of Y, when all X_k are known.
```

2.2 cEn_calcConditionalShannonFT.m

```
% function bits = ...
%   cEn_calcConditionalShannonFT( ftdata, chanlist, bins, exparams )
%
```

```
% This calculates the conditional entropy associated with a set of signals.
% This is the average amount of additional information that a sample from
% variable Y gives you when you already know variables X_1..X_k.
%
% This processes Field Trip data as input, concatenating trials.
%
% This needs a large number of samples to generate accurate results. To
% compensate for smaller sample counts, this may optionally use the
% extrapolation method described in EXTRAPOLATION.txt.
%
% "ftdata" is a ft_datatype_raw data structure produced by Field Trip.
% "chanlist" is a cell array containing channel labels or a vector containing
%   channel indices. If the list is empty, all channels are used. The first
%   channel in this list is used as the variable Y; remaining channels are
%   X_k.
% "bins" is a scalar or vector (to generate histogram bins) or a cell array
%   (to supply bin edge lists). If it's a vector of length Nchans or a
%   scalar, it indicates how many bins to use for each channel's data. If
%   it's a cell array, bins{chanidx} provides the list of edges used for
%   binning each channel's data.
% "exparams" is an optional structure containing extrapolation tuning
%   parameters, per EXTRAPOLATION.txt. If this is empty, default parameters
%   are used. If this is absent, no extrapolation is performed.
%
% "bits" is a scalar with the average additional entropy provided by an
%   observation of Y, when all X_k are known.
```

2.3 cEn_calcConditionalShannonHist.m

```
% function bits = cEn_calcConditionalShannonHist( bincounts )
%
% This calculates the conditional entropy associated with a multidimensional
% histogram. This is the average amount of additional information that a
% sample from variable Y gives you when you already know variable X.
%
% NOTE - This needs a large number of samples to generate accurate results!
%
% Conditional entropy is:
%   H(Y|X) = - sum_j,k[  P(x_j,y_k) * log2( P(x_j,y_k) / P(x_j) )  ]
%
% For independent variables, where P(x_j,y_k) = P(x_j) * P(y_k), this
% reduces to the Shannon entropy H(Y) (since X provides no information
% about Y).
%
% "bincounts" is a matrix with at least 2 dimensions containing histogram
%   counts. The first dimension corresponds to the variable Y. Remaining
%   dimensions are one or more X variables.
%
```

```
% "bits" is a scalar with the average additional entropy provided by an
%   observation of Y, when X is known.
```

2.4 cEn_calcExtrapWrapper.m

```
% function outval = cEn_calcExtrapWrapper( dataseries, datafunc, exparams )
%
% This calls a user-supplied function to process data (typically an
% entropy-calculation function). Results at low sample counts are
% extrapolated to estimate the result at large sample counts.
%
% Extrapolation is performed using the method described in Palmigiano 2017,
% which is based on the method described in Strong 1998, which is based on
% the analysis performed in Treves 1995:
%
% (Palmigiano 2017) "Flexible Information Routing by Transient Synchrony"
% (Strong 1998) "Entropy and Information in Neural Spike Trains"
% (Treves 1995) "The Upward Bias in Measures of Information Derived From
%   Limited Data Samples"
%
% For several values of N, M subsets of size (nsamples/N) are selected from
% the sample data. The entropy calculation is run on these M subsets and the
% results averaged, producing a result for the chosen value of N. A
% A quadratic fit is performed to the results (as a function of N), and the
% Y intercept (N = 0) is taken as the estimated infinite-length output.
%
% NOTE - If there are fewer than three divisors, the order of the curve fit
% will be reduced. A noteworthy use of this is divisors = [ 1 ], testcount
% = 1 to return the calculation result without extrapolation.
%
% "dataseries" is a (Nchans,Nsamples) matrix containing several data series.
% "datafunc" is a function handle, accepting one argument (the data series).
%   This function has the form:
%     function outval = datafunc( dataseries )
%   This is typically defined as a wrapper passing additional arguments:
%     datafunc = @( dataseries ) doSomething( dataseries, extra_args );
% "exparams" is a structure containing extrapolation parameters. This may
%   be empty; missing parameter values are set to default values. Fields are:
%   "divisors" is a vector containing divisors for nsamples. Default: [1:10]
%   "testcount" is the number of tests to average for each divisor. Default: 3
%
% "outval" is the extrapolated output of "datafunc".
```

2.5 cEn_calcLaggedMutualInfo.m

```
% function milist = ...
```

```
%   cEn_calcLaggedMutualInfo( dataseries, laglist, bins, exparams)
%
% This calculates the mutual information between a destination signal and
% time-lagged source signals. This is the amount of information shared
% between the variables, as measured by comparing the joint probability
% distribution with the distribution expected for independent variables.
%
% This is less informative than transfer entropy but can be faster to
% calculate.
%
% This needs a large number of samples to generate accurate results. To
% compensate for samller sample counts, this may optionally use the
% extrapolation method described in EXTRAPOLATION.txt.
%
% "dataseries" is a Nchans x Nsamples matrix or a cell array of length
%   Nchans containing data series. The first series (chan = 1) is the
%   destination signal; remaining series are source signals. For cell
%   array data, each cell contains either a vector of length Nsamples or
%   a matrix of size Ntrials x Nsamples.
% "laglist" is a vector containing sample time lags to test. These are
%   applied to the "source" signals. These may be negative (future times).
% "bins" is a scalar or vector (to generate bins) or a cell array (to supply
%   bin definitions). If it's a vector of length Nchans or a scalar, it
%   indicates how many bins to use for each channel's data. If it's a cell
%   array, bins{chanidx} provides the list of edges used for binning each
%   channel's data.
% "exparams" is an optional structure containing extrapolation tuning
%   parameters, per EXTRAPOLATION.txt. If this is empty, default parameters
%   are used. If this is absent, no extrapolation is performed.
%
% "milist" is a vector with the same dimensions as laglist containing
%   mutual information estimates for each time lag.
```

2.6 cEn_calcLaggedMutualInfoFT.m

```
% function milist = ...
%   cEn_calcLaggedMutualInfoFT( ftdata, chanlist, laglist, bins, exparams)
%
% This calculates the mutual information between a destination signal and
% time-lagged source signals. This is the amount of information shared
% between the variables, as measured by comparing the joint probability
% distribution with the distribution expected for independent variables.
%
% This is less informative than transfer entropy but can be faster to
% calculate.
%
% This processes Field Trip input, concatenating trials (after shifting).
%
```

```
% This needs a large number of samples to generate accurate results. To
% compensate for samller sample counts, this may optionally use the
% extrapolation method described in EXTRAPOLATION.txt.
%
% "ftdata" is a ft_datatype_raw structure produced by Field Trip.
% "chanlist" is a vector or cell array of length Nchans specifying which
%   channels to process. If this is a cell array, it contains Field Trip
%   channel labels. If this is a vector, it contains channel indices. The
%   first channel (element 1) is the destination signal; remaining channels
%   are source signals.
% "laglist" is a vector containing sample time lags to test. These are
%   applied to the "source" signals. These may be negative (future times).
% "bins" is a scalar or vector (to generate bins) or a cell array (to supply
%   bin definitions). If it's a vector of length Nchans or a scalar, it
%   indicates how many bins to use for each channel's data. If it's a cell
%   array, bins{chanidx} provides the list of edges used for binning each
%   channel's data.
% "exparams" is an optional structure containing extrapolation tuning
%   parameters, per EXTRAPOLATION.txt. If this is empty, default parameters
%   are used. If this is absent, no extrapolation is performed.
%
% "milist" is a vector with the same dimensions as laglist containing
%   mutual information estimates for each time lag.
```

2.7 cEn_calcLaggedMutualInfoFT_MT.m

```
% function milist = ...
%   cEn_calcLaggedMutualInfoFT_MT( ftdata, chanlist, laglist, bins, exparams)
%
% This calculates the mutual information between a destination signal and
% time-lagged source signals. This is the amount of information shared
% between the variables, as measured by comparing the joint probability
% distribution with the distribution expected for independent variables.
%
% This is less informative than transfer entropy but can be faster to
% calculate.
%
% This processes Field Trip input, concatenating trials (after shifting).
%
% This tests different lags in parallel with each other. This requires the
% Parallel Computing Toolbox.
%
% This needs a large number of samples to generate accurate results. To
% compensate for samller sample counts, this may optionally use the
% extrapolation method described in EXTRAPOLATION.txt.
%
% "ftdata" is a ft_datatype_raw structure produced by Field Trip.
% "chanlist" is a vector or cell array of length Nchans specifying which
```

```
%   channels to process. If this is a cell array, it contains Field Trip
%   channel labels. If this is a vector, it contains channel indices. The
%   first channel (element 1) is the destination signal; remaining channels
%   are source signals.
% "laglist" is a vector containing sample time lags to test. These are
%   applied to the "source" signals. These may be negative (future times).
% "bins" is a scalar or vector (to generate bins) or a cell array (to supply
%   bin definitions). If it's a vector of length Nchans or a scalar, it
%   indicates how many bins to use for each channel's data. If it's a cell
%   array, bins{chanidx} provides the list of edges used for binning each
%   channel's data.
% "exparams" is an optional structure containing extrapolation tuning
%   parameters, per EXTRAPOLATION.txt. If this is empty, default parameters
%   are used. If this is absent, no extrapolation is performed.
%
% "milist" is a vector with the same dimensions as laglist containing
%   mutual information estimates for each time lag.
```

2.8 cEn_calcLaggedMutualInfo_MT.m

```
% function milist = ...
%   cEn_calcLaggedMutualInfo_MT( dataseries, laglist, bins, exparams)
%
% This calculates the mutual information between a destination signal and
% time-lagged source signals. This is the amount of information shared
% between the variables, as measured by comparing the joint probability
% distribution with the distribution expected for independent variables.
%
% This is less informative than transfer entropy but can be faster to
% calculate.
%
% This tests different lags in parallel with each other. This requires the
% Parallel Computing Toolbox.
%
% This needs a large number of samples to generate accurate results. To
% compensate for samller sample counts, this may optionally use the
% extrapolation method described in EXTRAPOLATION.txt.
%
% "dataseries" is a Nchans x Nsamples matrix or a cell array of length
%   Nchans containing data series. The first series (chan = 1) is the
%   destination signal; remaining series are source signals. For cell
%   array data, each cell contains either a vector of length Nsamples or
%   a matrix of size Ntrials x Nsamples.
% "laglist" is a vector containing sample time lags to test. These are
%   applied to the "source" signals. These may be negative (future times).
% "bins" is a scalar or vector (to generate bins) or a cell array (to supply
%   bin definitions). If it's a vector of length Nchans or a scalar, it
%   indicates how many bins to use for each channel's data. If it's a cell
```

```
%   array, bins{chanidx} provides the list of edges used for binning each
%   channel's data.
% "exparams" is an optional structure containing extrapolation tuning
%   parameters, per EXTRAPOLATION.txt. If this is empty, default parameters
%   are used. If this is absent, no extrapolation is performed.
%
% "milist" is a vector with the same dimensions as laglist containing
%   mutual information estimates for each time lag.
```

2.9 cEn_calcMutualInfo.m

```
% function bits = cEn_calcMutualInfo( dataseries, bins, exparams )
%
% This calculates the mutual information associated with a set of signals.
% This is the amount of information shared between the variables, as
% measured by comparing the joint probability distribution with the
% distribution expected if the variables were independent.
%
% This needs a large number of samples to generate accurate results. To
% compensate for smaller sample counts, this may optionally use the
% extrapolation method described in EXTRAPOLATION.txt.
%
% "dataseries" is a (Nchans,Nsamples) matrix containing several data series.
% "bins" is a scalar or vector (to generate bins) or a cell array (to supply
%   bin definitions). If it's a vector of length Nchans or a scalar, it
%   indicates how many bins to use for each channel's data. If it's a cell
%   array, bins{chanidx} provides the list of edges used for binning each
%   channel's data.
% "exparams" is an optional structure containing extrapolation tuning
%   parameters, per EXTRAPOLATION.txt. If this is empty, default parameters
%   are used. If this is absent, no extrapolation is performed.
%
% "bits" is a scalar with the mutual information, computed as the reduction
%   in information content vs the joint distribution of independent variables.
```

2.10 cEn_calcMutualInfoFT.m

```
% function bits = cEn_calcMutualInfoFT( ftdata, chanlist, bins, exparams )
%
% This calculates the mutual information associated with a set of signals.
% This is the amount of information shared between the variables, as
% measured by comparing the joint probability distribution with the
% distribution expected if the variables were independent.
%
% This processes Field Trip data as input, concatenating trials.
%
```

```
% This needs a large number of samples to generate accurate results. To
% compensate for smaller sample counts, this may optionally use the
% extrapolation method described in EXTRAPOLATION.txt.
%
% "ftdata" is a ft_datatype_raw data structure produced by Field Trip.
% "chanlist" is a cell array containing channel labels or a vector containing
%   channel indices. If the list is empty, all channels are used.
% "bins" is a scalar or vector (to generate bins) or a cell array (to supply
%   bin definitions). If it's a vector of length Nchans or a scalar, it
%   indicates how many bins to use for each channel's data. If it's a cell
%   array, bins{chanidx} provides the list of edges used for binning each
%   channel's data.
% "exparams" is an optional structure containing extrapolation tuning
%   parameters, per EXTRAPOLATION.txt. If this is empty, default parameters
%   are used. If this is absent, no extrapolation is performed.
%
% "bits" is a scalar with the mutual information, computed as the reduction
%   in information content vs the joint distribution of independent variables.
```

2.11 cEn_calcMutualInfoHist.m

```
% function bits = cEn_calcMutualInfoHist( bincounts )
%
% This calculates the mutual information associated with a multidimensional
% histogram. This is the amount of information shared between the variables,
% as measured by comparing the joint probability distribution with the
% distribution expected if the variables were independent.
%
% NOTE - This needs a large number of samples to generate accurate results!
%
% Mutual information is:
%   I(X,Y) = sum_j,k[  P(x_j,y_k) * log2( P(x_j,y_k) / P(x_j) P(y_k) )  ]
%
% For independent variables, where P(x_j,y_k) = P(x_j) * P(y_k), this
% is zero bits (since there is no shared information between X and Y).
%
% For more than two variables, this function computes MI using:
%   P(x_j, y_k, ...) / ( P(x_j) * P(y_k) * ... )
%
% "bincounts" is a matrix with at least 2 dimensions containing histogram
%   counts.
%
% "bits" is a scalar with the mutual information, computed as the reduction
%   in information content vs the joint distribution of independent variables.
```

2.12 cEn_calcShannon.m

```
% function bits = cEn_calcShannon( dataseries, bins )
%
% This calculates the total Shannon entropy associated with a signal.
% This is the average number of bits of information that you get from seeing
% an observation.
%
% Shannon entropy is:   H(X) = - sum_k[ P(x_k) * log2(P(x_k)) ]
% (This is the probability-weighted average of the individual bin entropies.)
%
% A histogram is built of the input signal values, with each histogram bin
% corresponding to an input symbol, and the histogram being the probability
% distribution across symbols.
%
% "dataseries" is the signal to evaluate.
% "bins" is a scalar (to generate uniform histogram bins) or a vector
%   (to specify histogram bin edges).
%
% "bits" is a scalar with the average Shannon entropy of an observation.
```

2.13 cEn_calcShannonHist.m

```
% function bits = cEn_calcShannonHist( bincounts )
%
% This calculates the total Shannon entropy associated with a histogram.
% This is the average number of bits of information that you get from seeing
% an observation.
%
% Shannon entropy is:   H(X) = - sum_k[ P(x_k) * log2(P(x_k)) ]
% (This is the probability-weighted average of the individual bin entropies.)
%
% "bincounts" is a vector or matrix containing histogram counts.
%
% "bits" is a scalar with the average Shannon entropy of an observation.
```

2.14 cEn_calcShannonHistEach.m

```
% function binbits = cEn_calcShannonHistEach( bincounts )
%
% This calculates the Shannon entropy associated with each bin in a
% histogram. This is the number of bits of information that you get from
% seeing an observation that was in that bin.
%
% Shannon entropy is:   H(x_k) = - log2( P(x_k) )
```

```
%
% "bincounts" is a vector or matrix containing histogram counts.
%
% "binbits" is a matrix with the same dimensions as "bincounts" that
%   contains the Shannon entropy corresponding to each bin.
```

2.15 cEn_calcTransferEntropy.m

```
% function telist = ...
%   cEn_calcTransferEntropy( dataseries, laglist, bins, exparams )
%
% This calculates the partial transfer entropy from signals X_1..X_k to
% signal Y, for a specified set of time lags. If there is only one X, this
% is the transfer entropy from X to Y.
%
% NOTE - This needs a large number of samples to generate accurate results!
% To compensate for smaller sample counts, this may optionally use the
% extrapolation method described in EXTRAPOLATION.txt (per Palmigiano 2017).
%
% Transfer entropy from X to Y is defined as:
%   TEx->y = H[Y|Ypast] - H[Y|Ypast,Xpast]
% This is the amount of additional information gained about the future of Y
% by knowing the past of X, vs just knowing the past of Y.
%
% Partial transfer entropy from A to Y in the presence of B is defined as:
%   pTEa->y = H[Y|Ypast,Bpast] - H[Y|Ypast,Bpast,Apast]
% This is the amount of additional information gained about the future of Y
% by knowing the past of A, vs just knowing the past of Y and B.
%
% This is prohibitively expensive to compute, so a proxy is usually used
% that considers a sample at some distance in the past as a proxy for the
% entire past history:
%   TEx->y(tau) = H[Y(t)|Y(t-tau)] - H[Y(t)|Y(t-tau),X(t-tau)]
%
% A similar proxy is used for computing partial transfer entropy.
%
% NOTE - For k source signals, this involves evaluating a (k+2) dimensional
% histogram. This gets very big very quickly, and also needs a very large
% number of samples to get good statistics.
%
% "dataseries" is a Nchans x Nsamples matrix or a cell array of length
%   Nchans containing data series. The first series (chan = 1) is the
%   destination signal Y; remaining series are source signals X_k. For cell
%   array data, each cell contains either a vector of length Nsamples or
%   a matrix of size Ntrials x Nsamples.
% "laglist" is a vector containing sample lags to test. These correspond to
%   tau in the equation above. These may be negative (looking at the future).
% "bins" is a scalar or vector (to generate bins) or a cell array (to supply
```

```
%   bin definitions). If it's a vector of length Nchans or a scalar, it
%   indicates how many bins to use for each channel's data. If it's a cell
%   array, bins{chanidx} provides the list of edges used for binning each
%   channel's data.
% "exparams" is an optional structure containing extrapolation tuning
%   parameters, per EXTRAPOLATION.txt. If this is empty, default parameters
%   are used. If this is absent, no extrapolation is performed.
%
% "telist" is a (Nchans-1,Nlags) matrix containing transfer entropy
%   estimates from X_k (dataseries{k+1}) to Y (dataseries{1}) for each
%   time lag.
```

2.16 cEn_calcTransferEntropyFT.m

```
% function telist = ...
%   cEn_calcTransferEntropyFT( ftdata, chanlist, laglist, bins, exparams )
%
% This calculates the partial transfer entropy from signals X_1..X_k to
% signal Y, for a specified set of time lags. If there is only one X, this
% is the transfer entropy from X to Y.
%
% This processes Field Trip input, concatenating trials (after shifting).
%
% NOTE - This needs a large number of samples to generate accurate results!
% To compensate for smaller sample counts, this may optionally use the
% extrapolation method described in EXTRAPOLATION.txt (per Palmigiano 2017).
%
% Transfer entropy from X to Y is defined as:
%   TEx->y = H[Y|Ypast] - H[Y|Ypast,Xpast]
% This is the amount of additional information gained about the future of Y
% by knowing the past of X, vs just knowing the past of Y.
%
% Partial transfer entropy from A to Y in the presence of B is defined as:
%   pTEa->y = H[Y|Ypast,Bpast] - H[Y|Ypast,Bpast,Apast]
% This is the amount of additional information gained about the future of Y
% by knowing the past of A, vs just knowing the past of Y and B.
%
% This is prohibitively expensive to compute, so a proxy is usually used
% that considers a sample at some distance in the past as a proxy for the
% entire past history:
%   TEx->y(tau) = H[Y(t)|Y(t-tau)] - H[Y(t)|Y(t-tau),X(t-tau)]
%
% A similar proxy is used for computing partial transfer entropy.
%
% NOTE - For k source signals, this involves evaluating a (k+2) dimensional
% histogram. This gets very big very quickly, and also needs a very large
% number of samples to get good statistics.
%
```

```
% "ftdata" is a ft_datatype_raw structure produced by Field Trip.
% "chanlist" is a vector or cell array of length Nchans specifying which
%   channels to processes. If this is a cell array, it contains Field Trip
%   channel labels. If this is a vector, it contains channel indices. The
%   first channel (element 1) is the variable Y; remaining channels are X_k.
% "laglist" is a vector containing sample lags to test. These correspond to
%   tau in the equation above. These may be negative (looking at the future).
% "bins" is a scalar or vector (to generate bins) or a cell array (to supply
%   bin definitions). If it's a vector of length Nchans or a scalar, it
%   indicates how many bins to use for each channel's data. If it's a cell
%   array, bins{chanidx} provides the list of edges used for binning each
%   channel's data.
% "exparams" is an optional structure containing extrapolation tuning
%   parameters, per EXTRAPOLATION.txt. If this is empty, default parameters
%   are used. If this is absent, no extrapolation is performed.
%
% "telist" is a (Nchans-1,Nlags) matrix containing transfer entropy
%   estimates from X_k (dataseries{k+1}) to Y (dataseries{1}) for each
%   time lag.
```

2.17 cEn_calcTransferEntropyFT_MT.m

```
% function telist = ...
%   cEn_calcTransferEntropyFT_MT( ftdata, chanlist, laglist, bins, exparams )
%
% This calculates the partial transfer entropy from signals X_1..X_k to
% signal Y, for a specified set of time lags. If there is only one X, this
% is the transfer entropy from X to Y.
%
% This processes Field Trip input, concatenating trials (after shifting).
%
% This calls cEn_calcTransferEntropy_MT(), which tests different lags in
% parallel with each other. This requires the Parallel Computing Toolbox.
%
% NOTE - This needs a large number of samples to generate accurate results!
% To compensate for smaller sample counts, this may optionally use the
% extrapolation method described in EXTRAPOLATION.txt (per Palmigiano 2017).
%
% Transfer entropy from X to Y is defined as:
%   TEx->y = H[Y|Ypast] - H[Y|Ypast,Xpast]
% This is the amount of additional information gained about the future of Y
% by knowing the past of X, vs just knowing the past of Y.
%
% Partial transfer entropy from A to Y in the presence of B is defined as:
%   pTEa->y = H[Y|Ypast,Bpast] - H[Y|Ypast,Bpast,Apast]
% This is the amount of additional information gained about the future of Y
% by knowing the past of A, vs just knowing the past of Y and B.
%
```

```
% This is prohibitively expensive to compute, so a proxy is usually used
% that considers a sample at some distance in the past as a proxy for the
% entire past history:
%   TEx->y(tau) = H[Y(t)|Y(t-tau)] - H[Y(t)|Y(t-tau),X(t-tau)]
%
% A similar proxy is used for computing partial transfer entropy.
%
% NOTE - For k source signals, this involves evaluating a (k+2) dimensional
% histogram. This gets very big very quickly, and also needs a very large
% number of samples to get good statistics.
%
% "ftdata" is a ft_datatype_raw structure produced by Field Trip.
% "chanlist" is a vector or cell array of length Nchans specifying which
%   channels to processes. If this is a cell array, it contains Field Trip
%   channel labels. If this is a vector, it contains channel indices. The
%   first channel (element 1) is the variable Y; remaining channels are X_k.
% "laglist" is a vector containing sample lags to test. These correspond to
%   tau in the equation above. These may be negative (looking at the future).
% "bins" is a scalar or vector (to generate bins) or a cell array (to supply
%   bin definitions). If it's a vector of length Nchans or a scalar, it
%   indicates how many bins to use for each channel's data. If it's a cell
%   array, bins{chanidx} provides the list of edges used for binning each
%   channel's data.
% "exparams" is an optional structure containing extrapolation tuning
%   parameters, per EXTRAPOLATION.txt. If this is empty, default parameters
%   are used. If this is absent, no extrapolation is performed.
%
% "telist" is a (Nchans-1,Nlags) matrix containing transfer entropy
%   estimates from X_k (dataseries{k+1}) to Y (dataseries{1}) for each
%   time lag.
```

2.18 cEn_calcTransferEntropy_MT.m

```
% function telist = ...
%   cEn_calcTransferEntropy_MT( dataseries, laglist, bins, exparams )
%
% This is a wrapper for cEn_calcTransferEntropy() that tests different lags
% in parallel with each other. This requires the Parallel Computing Toolbox.
%
% This calculates the partial transfer entropy from signals X_1..X_k to
% signal Y, for a specified set of time lags. If there is only one X, this
% is the transfer entropy from X to Y.
%
% NOTE - This needs a large number of samples to generate accurate results!
% To compensate for smaller sample counts, this may optionally use the
% extrapolation method described in EXTRAPOLATION.txt (per Palmigiano 2017).
%
% Transfer entropy from X to Y is defined as:
```

```
%   TEx->y = H[Y|Ypast] - H[Y|Ypast,Xpast]
% This is the amount of additional information gained about the future of Y
% by knowing the past of X, vs just knowing the past of Y.
%
% Partial transfer entropy from A to Y in the presence of B is defined as:
%   pTEa->y = H[Y|Ypast,Bpast] - H[Y|Ypast,Bpast,Apast]
% This is the amount of additional information gained about the future of Y
% by knowing the past of A, vs just knowing the past of Y and B.
%
% This is prohibitively expensive to compute, so a proxy is usually used
% that considers a sample at some distance in the past as a proxy for the
% entire past history:
%   TEx->y(tau) = H[Y(t)|Y(t-tau)] - H[Y(t)|Y(t-tau),X(t-tau)]
%
% A similar proxy is used for computing partial transfer entropy.
%
% NOTE - For k source signals, this involves evaluating a (k+2) dimensional
% histogram. This gets very big very quickly, and also needs a very large
% number of samples to get good statistics.
%
% "dataseries" is a Nchans x Nsamples matrix or a cell array of length
%   Nchans containing data series. The first series (chan = 1) is the
%   destination signal Y; remaining series are source signals X_k. For cell
%   array data, each cell contains either a vector of length Nsamples or
%   a matrix of size Ntrials x Nsamples.
% "laglist" is a vector containing sample lags to test. These correspond to
%   tau in the equation above. These may be negative (looking at the future).
% "bins" is a scalar or vector (to generate bins) or a cell array (to supply
%   bin definitions). If it's a vector of length Nchans or a scalar, it
%   indicates how many bins to use for each channel's data. If it's a cell
%   array, bins{chanidx} provides the list of edges used for binning each
%   channel's data.
% "exparams" is an optional structure containing extrapolation tuning
%   parameters, per EXTRAPOLATION.txt. If this is empty, default parameters
%   are used. If this is absent, no extrapolation is performed.
%
% "telist" is a (Nchans-1,Nlags) matrix containing transfer entropy
%   estimates from X_k (dataseries{k+1}) to Y (dataseries{1}) for each
%   time lag.
```

2.19 cEn_extrapFillParams.m

```
% function newparams = cEn_extrapFillParams( oldparams )
%
% This function fills in missing fields in extrapolation parameter
% structures used by cEn_calcExtrapWrapper(), per EXTRAPOLATION.txt.
%
% "oldparams" is an extrapolation parameter structure to modify.
```

```
%
% "newparams" is a copy of "oldparams" with missing fields filled in.
```

2.20 cEn_ftHelperChannelToMatrix.m

```
% function dataseries = cEn_ftHelperChannelToMatrix( ftdata, chanwanted )
%
% This extracts one channel's trials from a ft_datatype_raw dataset and
% converts it to a Ntrials x Nsamples data matrix.
%
% "ftdata" is a ft_datatype_raw structure containing trial data.
% "chanwanted" is a character vector containing a channel label, or a
%   scalar containing a channel index. If this is empty or invalid, the
%   first channel in ftdata.label is chosen.
%
% "dataseries" is a Ntrials x Nsamples matrix containing the desired
%   channel's trial data.
```

2.21 cEn_ftHelperCheckChannels.m

```
% function chanindices = cEn_ftHelperCheckChannels( ftlabels, chanlist )
%
% This checks a user-supplied list of channels or channel indices against
% a Field Trip channel list, translating and adjusting as needed.
%
% Invalid requested channels result in indices of NaN.
%
% "ftlabels" is the "label" cell array from a ft_datatype_raw structure.
% "chanlist" is a cell array containing channel labels or a vector containing
%   channel indices or a character vector containing a single channel label.
%   Channels are processed in the order given in this list. If the list is
%   empty, "ftlabels" is used as the list.
%
% "chanindices" is a vector of the same size as "chanlist" containing
%   channel indices of the requested channels, in the order specified by
%   "chanlist". Invalid requested channels have indices of NaN.
```

2.22 cEn_ftHelperConcatTrials.m

```
% function dataseries = cEn_ftHelperConcatTrials( ftdata, chanlist )
%
% This extracts trial data from a ft_datatype_raw dataset, concatenates
% the trials, and returns a Nchans x Nsamples data matrix with selected
```

```
% channels.
%
% "ftdata" is a ft_datatype_raw structure containing trial data.
% "chanlist" is a cell array containing channel labels or a vector containing
%   channel indices. Channels are assembled in the order given in this list.
%   If the list is empty, ftdata.label is used as the list.
%
% "dataseries" is a Nchans x Nsamples matrix containing concatenated trial
%   data. Channels are copied in the order specified by "chanlist".
```

2.23 cEn_getBinnedMultivariate.m

```
% function [ bincounts edges ] = cEn_getBinnedMultivariate( dataseries, bins )
%
% This tallies multidimensional histogram counts representing the joint
% probability distribution between several data series. Histogram bins may
% either be generated or be supplied by the caller.
%
% If histogram bins are generated, they're chosen such that if each series
% were binned in isolation, each bin would have an approximately equal
% number of samples.
%
% NOTE - High-dimensional histograms can get very big very fast!
%
% "dataseries" is a (Nchans,Nsamples) matrix containing several data series.
% "bins" is a scalar or vector (to generate bins) or a cell array (to supply
%   bin definitions). If it's a vector of length Nchans or a scalar, it
%   indicates how many bins to use for each channel's data. If it's a cell
%   array, bins{chanidx} provides the list of edges used for binning each
%   channel's data.
%
% "bincounts" is a multidimensional matrix with histogram bin counts. The
%   k-th dimension corresponds to dataseries(k,:) and has length numbins(k).
% "edges" {chanidx} is a cell array, with each cell containing a vector with
%   bin edges for one channel's data.
```

2.24 cEn_getHistBinsDiscrete.m

```
% function edges = cEn_getHistBinsEqPop( datavals )
%
% This finds a set of histogram bin edges such that each unique data value
% in the input series has an associated bin.
%
% This is intended to be used with discrete-valued data (such as spike
% counts).
%
```

```
% "datavals" is a vector or matrix containing samples to be binned.
%
% "edges" is a vector containing bin edges.
```

2.25 cEn_getHistBinsEqPop.m

```
% function edges = cEn_getHistBinsEqPop( datavals, numbins )
%
% This finds a set of histogram bin edges that results in having
% approximately equal numbers of samples in each bin.
%
% "datavals" is a vector or matrix containing samples to be binned.
% "numbins" is the number of bins.
%
% "edges" is a vector of length (numbins+1) containing bin edges.
```

2.26 cEn_getMultivariateHistBins.m

```
% function edges = cEn_getMultivariateHistBins( dataseries, numbins )
%
% This finds a set of histogram bins for multivariate data such that if
% each variable were binned in isolation, each bin would have an
% approximately equal number of samples.
%
% "dataseries" is a (Nchans,Nsamples) matrix containing several data series.
% "numbins" is either a vector of length Nchans or a scalar, indicating how
%   many histogram bins to use for each channel's data.
%
% "edges" {chanidx} is a cell array, with each cell containing a vector with
%   bin edges for one channel's data.
```

2.27 cEn_getMultivariateHistBinsDiscrete.m

```
% function edges = cEn_getMultivariateHistBinsDiscrete( dataseries )
%
% This finds a set of histogram bins for multivariate data such that if
% each variable were binned in isolation, each unique data value for that
% variable would have an associated bin.
%
% "dataseries" is a (Nchans,Nsamples) matrix containing several data series.
%
% "edges" {chanidx} is a cell array, with each cell containing a vector with
%   bin edges for one channel's data.
```

2.28 cEn_getMultivariateHistBinsDiscreteFT.m

```
% function edges = cEn_getMultivariateHistBinsDiscreteFT( ftdata, chanlist )
%
% This finds a set of histogram bins for multivariate data such that if
% each variable were binned in isolation, each unique data value for that
% variable would have an associated bin.
%
% This processes Field Trip input.
%
% "ftdata" is a ft_datatype_raw structure produced by Field Trip.
% "chanlist" is a vector or cell array of length Nchans specifying which
%   channels to process. If this is a cell array, it contains Field Trip
%   channel labels. If this is a vector, it contains channel indices.
%
% "edges" {chanidx} is a cell array, with each cell containing a vector with
%   bin edges for one channel's data.
```

2.29 cEn_teHelperShiftAndLinearize.m

```
% function [ dstpresent dstpast srcpast ] = ...
%   cEn_teHelperShiftAndLinearize( dstseries, srcseries, timelag, laglist )
%
% This produces cropped time-shfited series used for transfer entropy and
% partial transfer entropy calculations. Trials are concatenated after
% shifting and cropping.
%
% NOTE - Instead of shifting the "past" signals left, the "present" signal
% is shifted right (for positive time lags; reverse for negative).
%
% "dstseries" is a vector of length Nsamples or a Ntrials x Nsamples matrix
%   containing the destination signal Y.
% "srcseries" is a cell array of length Nsrcs containing source signals X_k.
%   These are either vectors of length Nsamples or matrices of size
%   Ntrials x Nsamples.
% "timelag" is the time lag to test, in samples.
% "laglist" is a vector containing all tested time lags (to get lag range).
%
% "dstpresent" is a 1xNoutsamps vector containing concatenated cropped
%   time-shifted trials from dstseries.
% "dstpast" is a 1xNoutsamps vector containing concatenated cropped trials
%   from dstseries.
% "srcpast" is a cell array of length Nsrcs containing 1xNoutsamps vectors
%   that are concatenated cropped trials from srcseries.
```

Chapter 3

Sample Code

3.1 do_demo.m

```
% Sample code for the entropy library.
% Written by Christopher Thomas.


%
% Configuration.

% This needs the Parallel Computing Toolbox.
want_parallel = false;

swept_histbins = [ 8 16 32 ];

% 30k takes a minute or two. Higher counts are more informative.
swept_sampcounts = [ 3000 10000 30000 ];
%swept_sampcounts = [ 1000 swept_sampcounts ];
%swept_sampcounts = [ swept_sampcounts 100000 ];
%swept_sampcounts = [ swept_sampcounts 300000 ];
%swept_sampcounts = [ swept_sampcounts 1000000 ];

laglist = [ -15:15 ];
test_lag = 6;

signal_plot_samps = 300;

plotdir = 'plots';



%
% Startup.
```

```
addpath('../library');

if want_parallel
  parpool;
end



%
% Compute mutual information, lagged mutual information, and transfer
% entropy.


% Initialize output data.
% We aggregate it and then plot as a separate step.

mutualbits_weak = ...
  nan([ length(swept_sampcounts), length(swept_histbins) ]);
mutualbits_strong = mutualbits_weak;
mutualbits_extrap = mutualbits_weak;
mutualbits_3ch = mutualbits_weak;

mutualbits_lagged_st = ...
  nan([ length(laglist), length(swept_sampcounts), length(swept_histbins) ]);
mutualbits_lagged_wk = mutualbits_lagged_st;
mutualbits_discrete = mutualbits_lagged_st;
mutualbits_discrete_auto = ...
  nan([ length(laglist), length(swept_sampcounts), 1 ]);

transferbits_st = mutualbits_lagged_st;
transferbits_wk = mutualbits_lagged_st;
transferbits_discrete = mutualbits_lagged_st;
transferbits_discrete_auto = mutualbits_discrete_auto;

ptebits_st = mutualbits_lagged_st;
ptebits_wk = mutualbits_lagged_st;


% Iterate sample counts.

for sidx = 1:length(swept_sampcounts)

  sampcount = swept_sampcounts(sidx);

  disp(sprintf('== Calculating statistics for %d samples.', sampcount));


  %
  % Build test signals.
```

```
signalData = helper_makeDataSignal( sampcount, 'sine' );
signalN1 = helper_makeDataSignal( sampcount, 'sine' );
signalN2 = helper_makeDataSignal( sampcount, 'sine' );
signalN3 = helper_makeDataSignal( sampcount, 'sine' );

signalY = 0.7 * signalData + 0.3 * signalN1;
signalXst = 0.7 * signalData + 0.3 * signalN2;
signalXwk = 0.4 * signalData + 0.6 * signalN3;

datamatrix_strong = [ signalY ; signalXst ];
datamatrix_weak = [ signalY ; signalXwk ];
datamatrix_3ch = [ signalY ; signalXst ; signalXwk ];

datamatrix_lagged_st = [ signalY ; circshift( signalXst, test_lag ) ];
datamatrix_lagged_wk = [ signalY ; circshift( signalXwk, test_lag ) ];
datamatrix_3ch_lagged = ...
  [ signalY ; circshift( signalXst, test_lag ) ; ...
    circshift( signalXwk, test_lag ) ];

discreteData = helper_makeDataSignal( sampcount, 'counts' );
discreteN1 = helper_makeDataSignal( sampcount, 'counts' );
discreteN2 = helper_makeDataSignal( sampcount, 'counts' );

discreteY = discreteData + discreteN1;
discreteX = discreteData + discreteN2;

datamatrix_discrete = [ discreteY ; circshift( discreteX, test_lag ) ];


% Plot the signals for one sample count, to show what they're like.

if 1 == sidx
  helper_plotDataSignals(datamatrix_strong, { 'Y', 'Xst' }, ...
    signal_plot_samps, 'Strongly Coupled Signals', ...
    [ plotdir filesep 'signals-strong.png' ] );

  helper_plotDataSignals(datamatrix_weak, { 'Y', 'Xwk' }, ...
    signal_plot_samps, 'Weakly Coupled Signals', ...
    [ plotdir filesep 'signals-weak.png' ] );

  helper_plotDataSignals(datamatrix_lagged_st, { 'Y', 'Xst' }, ...
    signal_plot_samps, 'Time-Lagged Signals', ...
    [ plotdir filesep 'signals-lagged.png' ] );

  helper_plotDataSignals(datamatrix_discrete, { 'Y', 'X' }, ...
    signal_plot_samps, 'Time-Lagged Discrete Signals (event counts)', ...
    [ plotdir filesep 'signals-discrete.png' ] );
end
```

```
% Set up the 3-channel cases as Field Trip structures, to demonstrate that.

ftdata_3ch = struct();
ftdata_3ch.fsample = 1000;
timeseries = 1:sampcount;
ftdata_3ch.time = { (timeseries - 1) / 1000 };
ftdata_3ch.label = { 'chY' ; 'chXst' ; 'chXwk' };
ftdata_3ch.trial = { datamatrix_3ch };

ftdata_3ch_lagged = ftdata_3ch;
ftdata_3ch_lagged.label = { 'chY' ; 'chXst' ; 'chXwk' };
ftdata_3ch_lagged.trial = { datamatrix_3ch_lagged };


%
% Compute mutual information.

tic;

for bidx = 1:length(swept_histbins)

  histbins = swept_histbins(bidx);

  % Calculate mutual information without extrapolation.

  mutualbits_weak( sidx, bidx ) = ...
    cEn_calcMutualInfo( datamatrix_weak, histbins );
  mutualbits_strong( sidx, bidx ) = ...
    cEn_calcMutualInfo( datamatrix_strong, histbins );


  % Calculate it again for the "weak" case, using extrapolation.
  % Do this by adding an extrapolation parameter structure as the last
  % argument. An empty structure gets filled with default settings.
  mutualbits_extrap( sidx, bidx ) = ...
    cEn_calcMutualInfo( datamatrix_weak, histbins, struct() );


  % Calculate 3-channel mutual information using a Field Trip structure.
  % No extrapolation.
  % Give it an empty channel list to say "use all channels".

  mutualbits_3ch( sidx, bidx ) = ...
    cEn_calcMutualInfoFT( ftdata_3ch, {}, histbins );

end

% Progress report.

durstring = helper_makePrettyTime(toc);
```

```
disp([ '.. Mutual information took ' durstring '.' ]);


%
% Compute time-lagged mutual information.
% This is similar to pairwise transfer entropy but is cheaper to compute.

tic;

for bidx = 1:length(swept_histbins)

  histbins = swept_histbins(bidx);

  % Don't use extrapolation for the demo.
  % Results are similar to above: It converges faster but isn't monotonic.

  if want_parallel
    mutualbits_lagged_st(:, sidx, bidx) = cEn_calcLaggedMutualInfo_MT( ...
      datamatrix_lagged_st, laglist, histbins );
    mutualbits_lagged_wk(:, sidx, bidx) = cEn_calcLaggedMutualInfo_MT( ...
      datamatrix_lagged_wk, laglist, histbins );

    mutualbits_discrete(:, sidx, bidx) = cEn_calcLaggedMutualInfo_MT( ...
      datamatrix_discrete, laglist, histbins );
  else
    mutualbits_lagged_st(:, sidx, bidx) = cEn_calcLaggedMutualInfo( ...
      datamatrix_lagged_st, laglist, histbins );
    mutualbits_lagged_wk(:, sidx, bidx) = cEn_calcLaggedMutualInfo( ...
      datamatrix_lagged_wk, laglist, histbins );

    mutualbits_discrete(:, sidx, bidx) = cEn_calcLaggedMutualInfo( ...
      datamatrix_discrete, laglist, histbins );
  end

end

% For discrete data (event counts), we have the option of using one bin
% per count value. There's a FT version of this function too.

histbins = cEn_getMultivariateHistBinsDiscrete( datamatrix_discrete );

% NOTE - Report the number of histogram bins.
% "histbins" is a cell array of vectors with bin edges.
thismsg = '.. Auto-detected bin counts for discrete data: ';
for bidx = 1:length(histbins)
  thismsg = [ thismsg sprintf('  %d', length(histbins{bidx}) - 1) ];
end
disp(thismsg);

if want_parallel
```

```
  mutualbits_discrete_auto(:,sidx,1) = cEn_calcLaggedMutualInfo_MT( ...
    datamatrix_discrete, laglist, histbins );
else
  mutualbits_discrete_auto(:,sidx,1) = cEn_calcLaggedMutualInfo( ...
    datamatrix_discrete, laglist, histbins );
end

% Progress report.

durstring = helper_makePrettyTime(toc);
disp([ '.. Lagged mutual information took ' durstring '.' ]);


%
% Compute two-channel transfer entropy.

tic;

for bidx = 1:length(swept_histbins)

  histbins = swept_histbins(bidx);

  % Don't use extrapolation for the demo.
  % Results are similar to above: It converges faster but isn't monotonic.

  if want_parallel
    transferbits_st(:, sidx, bidx) = cEn_calcTransferEntropy_MT( ...
      datamatrix_lagged_st, laglist, histbins );
    transferbits_wk(:, sidx, bidx) = cEn_calcTransferEntropy_MT( ...
      datamatrix_lagged_wk, laglist, histbins );

    transferbits_discrete(:, sidx, bidx) = cEn_calcTransferEntropy_MT( ...
      datamatrix_discrete, laglist, histbins );
  else
    transferbits_st(:, sidx, bidx) = cEn_calcTransferEntropy( ...
      datamatrix_lagged_st, laglist, histbins );
    transferbits_wk(:, sidx, bidx) = cEn_calcTransferEntropy( ...
      datamatrix_lagged_wk, laglist, histbins );

    transferbits_discrete(:, sidx, bidx) = cEn_calcTransferEntropy( ...
      datamatrix_discrete, laglist, histbins );
  end

end

% For discrete data (event counts), we have the option of using one bin
% per count value. There's a FT version of this function too.

histbins = cEn_getMultivariateHistBinsDiscrete( datamatrix_discrete );
```

```
  if want_parallel
    transferbits_discrete_auto(:,sidx,1) = cEn_calcTransferEntropy_MT( ...
      datamatrix_discrete, laglist, histbins );
  else
    transferbits_discrete_auto(:,sidx,1) = cEn_calcTransferEntropy( ...
      datamatrix_discrete, laglist, histbins );
  end

  % Progress report.

  durstring = helper_makePrettyTime(toc);
  disp([ '.. Two-channel transfer entropy took ' durstring '.' ]);


  %
  % Compute partial transfer entropy.

  tic;

  for bidx = 1:length(swept_histbins)

    histbins = swept_histbins(bidx);

    % Don't use extrapolation for the demo.
    % Results are similar to above: It converges faster but isn't monotonic.

    % Show how to do this with Field Trip data.
    % We can give a cell array of channel names or a vector of channel
    % indices. If we give {} or [], it means "use all channels".

    if want_parallel
      ptedata = cEn_calcTransferEntropyFT_MT( ...
        ftdata_3ch_lagged, { 'chY', 'chXst', 'chXwk' }, laglist, histbins );
      ptebits_st( :, sidx, bidx ) = ptedata(1,:);
      ptebits_wk( :, sidx, bidx ) = ptedata(2,:);
    else
      ptedata = cEn_calcTransferEntropyFT( ...
        ftdata_3ch_lagged, { 'chY', 'chXst', 'chXwk' }, laglist, histbins );
      ptebits_st( :, sidx, bidx ) = ptedata(1,:);
      ptebits_wk( :, sidx, bidx ) = ptedata(2,:);
    end

  end

  % Progress report.

  durstring = helper_makePrettyTime(toc);
  disp([ '.. Partial transfer entropy took ' durstring '.' ]);

end
```

```
disp('== Finished calculating statistics.');



%
% Plot the resulting mutual information and transfer entropy estimates.

disp('== Generating plots.');


% Mutual information.

helper_plotMutualInfo( ...
  mutualbits_weak, swept_sampcounts, swept_histbins, ...
  'Mutual Information (bits)', 'Mutual Information (weak coupling)', ...
  [ plotdir filesep 'mutual-weak.png' ] );

helper_plotMutualInfo( ...
  mutualbits_strong, swept_sampcounts, swept_histbins, ...
  'Mutual Information (bits)', 'Mutual Information (strong coupling)', ...
  [ plotdir filesep 'mutual-strong.png' ] );

helper_plotMutualInfo( ...
  mutualbits_extrap, swept_sampcounts, swept_histbins, ...
  'Mutual Information (bits)', ...
  'Mutual Information (weak coupling) (extrapolated)', ...
  [ plotdir filesep 'mutual-extrap.png' ] );

helper_plotMutualInfo( ...
  mutualbits_3ch, swept_sampcounts, swept_histbins, ...
  'Mutual Information (bits)', 'Mutual Information (three channels)', ...
  [ plotdir filesep 'mutual-three.png' ] );


% Time-lagged mutual information.

helper_plotLagged( ...
  mutualbits_lagged_st, ...
  laglist, test_lag, swept_sampcounts, swept_histbins, ...
  'Mutual Information (bits)', ...
  'Time-Lagged Mutual Information (strong channel)', ...
  [ plotdir filesep 'lagged-mutual-st-%s.png' ] );

helper_plotLagged( ...
  mutualbits_lagged_wk, ...
  laglist, test_lag, swept_sampcounts, swept_histbins, ...
  'Mutual Information (bits)', ...
  'Time-Lagged Mutual Information (weak channel)', ...
  [ plotdir filesep 'lagged-mutual-wk-%s.png' ] );
```

```
helper_plotLagged( ...
  mutualbits_discrete, ...
  laglist, test_lag, swept_sampcounts, swept_histbins, ...
  'Mutual Information (bits)', ...
  'Time-Lagged Mutual Information (discrete events)', ...
  [ plotdir filesep 'lagged-mutual-disc-%s.png' ] );

helper_plotLagged( ...
  mutualbits_discrete_auto, ...
  laglist, test_lag, swept_sampcounts, NaN, ...
  'Mutual Information (bits)', ...
  'Time-Lagged Mutual Information (discrete events)', ...
  [ plotdir filesep 'lagged-mutual-autodisc-%s.png' ] );


% Transfer entropy.

helper_plotLagged( ...
  transferbits_st, laglist, test_lag, swept_sampcounts, swept_histbins, ...
  'Transfer Entropy (bits)', 'Transfer Entropy (strong channel)', ...
  [ plotdir filesep 'transfer-st-%s.png' ], 'squashzero' );

helper_plotLagged( ...
  transferbits_wk, laglist, test_lag, swept_sampcounts, swept_histbins, ...
  'Transfer Entropy (bits)', 'Transfer Entropy (weak channel)', ...
  [ plotdir filesep 'transfer-wk-%s.png' ], 'squashzero' );

helper_plotLagged( ...
  transferbits_discrete, ...
  laglist, test_lag, swept_sampcounts, swept_histbins, ...
  'Transfer Entropy (bits)', 'Transfer Entropy (discrete events)', ...
  [ plotdir filesep 'transfer-disc-%s.png' ], 'squashzero' );

helper_plotLagged( ...
  transferbits_discrete_auto, ...
  laglist, test_lag, swept_sampcounts, NaN, ...
  'Transfer Entropy (bits)', 'Transfer Entropy (discrete events)', ...
  [ plotdir filesep 'transfer-autodisc-%s.png' ], 'squashzero' );


% Partial transfer entropy.

helper_plotLagged( ...
  ptebits_st, laglist, test_lag, swept_sampcounts, swept_histbins, ...
  'Transfer Entropy (bits)', 'Partial Transfer Entropy (strong channel)', ...
  [ plotdir filesep 'pte-st-%s.png' ], 'squashzero' );

helper_plotLagged( ...
  ptebits_wk, laglist, test_lag, swept_sampcounts, swept_histbins, ...
```

```
  'Transfer Entropy (bits)', 'Partial Transfer Entropy (weak channel)', ...
  [ plotdir filesep 'pte-wk-%s.png' ], 'squashzero' );


disp('== Finished generating plots.');



%
% This is the end of the file.
```