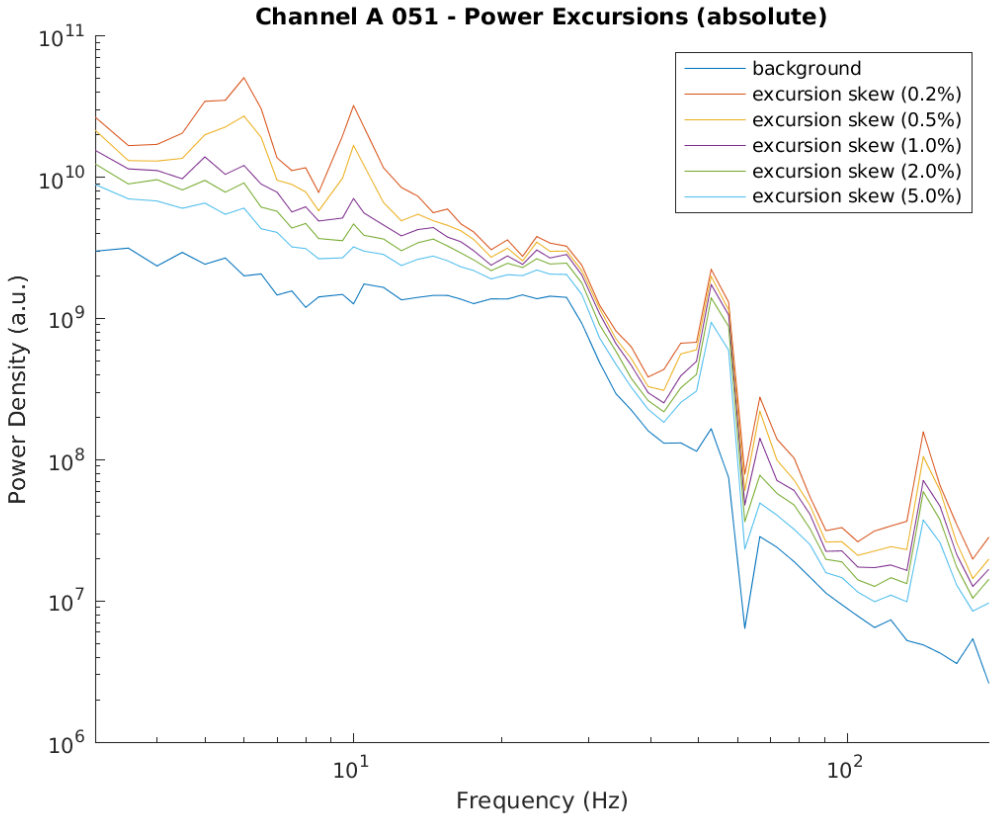
NeuroLoop Utilities – Function Reference

Written by Christopher Thomas – November 12, 2021.



Contents

Chapter 1

Overview

The NeuroLoop utility library functions are divided into several categories:

- **"Processing"** functions (ch. 3) perform signal processing operations on data series such as filtering, artifact rejection, statistical calculations, and so forth.
- **"Utility"** functions (ch. 4) perform miscellaneous operations that are not covered by the other categories.
- **"Plotting"** functions (ch. 5) render plots of various types to files, figures, or axes. These are included as an aid to rapid prototyping, and are used by the GUI scripts. The output is generally not publication-ready.
- **"I/O"** functions (ch. 6) facilitate operations for reading and writing data that aren't vendor-specific.
- **"Intan"** functions (ch. 7) facilitate the use of datasets stored in Intan's format. Only a subset of the feature set is presently supported.
- **"Channel Tool"** functions (ch. 8) perform operations used by the "Channel Tool" utility. The intention is that all operations that are not tied to GUI implementation are packaged as library functions for reuse outside of that application.

Several types of structure and several types of function handle are used by the library functions. These are described in Chapter 2.

FIXME: Most of these aren't documented yet!

Chapter 2

Special Structures and Function Handles

2.1 CHANLIST.txt

Lists of signal channels grouped by bank are saved in a "channel list"
structure. This structure has field names that are bank identifiers, with
each field containing an array of channel numbers.

Intan on-chip amplifiers have banks named "A" through "H" with channels 0-127.

FIXME - Chip auxiliary analog inputs not yet supported.

The Intan recording and stimulation controller auxiliary I/O ports have banks
named "AIN", "DIN", and "DOUT". "AIN" has channels 1-8 and "DIN" and "DOUT"
have channels 1-16.

FIXME - Controller analog outputs not yet supported (not saved?). These are
always duplicates of amplifier channels, so the data can still be accessed.

FIXME - Not including a time series field. Time series is a special case.

2.2 ZMODELS.txt

An impedance model is a list of category definitions for clustering impedance
values. This is stored as a structure indexed by category label, with each
field containing a structure that defines that category's cluster.

A "box" cluster accepts all data points within a given range of magnitudes
and phase angles.
A "box" cluster definition structure has the following fields:

"type" is 'box'.

"magrange" [min max] is the range of accepted impedance magnitudes. This is
  typically the logarithm of the actual magnitude.
"phaserange" [min max] is the range of accepted impedance phase angles, in
  radians. A pair of angles defines two arcs - one larger than pi radians,
  and one smaller. The smaller arc is taken to be the accepted range.

An "orthogauss" cluster definition is a bivariate normal distribution with
principal axes parallel to the magnitude and phase axes. The category label
of a sample is the category whose probability density function is highest
for that sample, out to some maximum range (typically 3 sigma).
An "orthogauss" cluster definition structure has the following fields:

"type" is 'orthogauss'.
"magmean" is the magnitude distribution's mean.
"magdev" is the magnitude distribution's standard deviation.
"phasemean" is the phase distribution's circular mean (mean direction).
"phasedev" is the standard deviation of (phase - phase mean). This is
  assumed to be much smaller than 2pi (not needing circular statistics).

FIXME - There's a planned "gauss" model that has proper multivariate
Gaussian distributions with covariance matrices, but impedance data rarely
actually needs that, so it's deferred.

This is the end of the file.

Chapter 3

"nlProc" Functions

3.1 nlProc_autoClusterImpedance.m

```
% function zmodels = ...
%   nlProc_autoClusterImpedance( magdata, phasedata, phaseunits )
%
% This attempts to build sensible cluster definitions covering the specified
% impedance magnitude and phase data. Cluster models are described in
% "ZMODELS.txt".
%
% NOTE - Magnitude input is in ohms and phase input may be radians or degrees,
% but model parameters use log10(ohms) and radians exclusively.
%
% "magdata" is a set of impedance magnitude values, in ohms.
% "phasedata" is a corresponding set of impedance phase values.
% "phaseunits" is 'degrees' or 'radians'.
%
% "zmodels" is a structure containing zero or more models of the data, indexed
%   by model type (typically 'box' and 'orthogauss').
%
% A 'box' model is a structure indexed by category label with the following
% fields:
%   "type" is 'box'.
%   "magrange" [min max] is the range of accepted log10(magnitude) values.
%   "phaserange" [min max] is the range of accepted phases in radians.
% A category label is applied to a sample if that sample's magnitude and
% phase are within the specified ranges.
%
% An 'orthogauss' model is a structure indexed by category label with the
% following fields:
%   "type" is 'orthogauss'.
%   "magmean" is the mean of log10(magnitude) for this category.
%   "magdev" is the standard deviation of log10(magnitude) for this category.
%   "phasemean" is the mean direction of phase for this category in radians.
```

```
%   "phasedev" is the standard deviation of (phase - mean) for this category.
% The category label of a sample is the category whose probability density
% function is highest for that sample.
```

3.2 nlProc_binTableDataSimple.m

```
% function newtable = ...
%  nlProc_binTableDataSimple( oldtable, bindefs, testorder, newcol )
%
% This applies catagory labels to table data rows by partitioning based on
% values in one or more table columns. Category labels are character arrays.
%
% "oldtable" is the table to apply labels to.
% "bindefs" is a structure indexed by category label. Each field contains
%   a structure array specifying conditions for that category label. A
%   condition structure contains the following fields:
%     "source" is the column to test.
%     "range" [min max] is the range of accepted values.
%     "negate" is true to only accept values _outside_ the range.
%   All conditions must match for the label to be applied. Tests on
%   non-existent source columns automatically fail.
% "testorder" is a cell array of category labels specifying the order in
%   which to test bin definitions. The last successful test determines the
%   label applied. Labels that don't have bin definitions automatically
%   succeed (this is useful for specifying a default label). Bin definitions
%   not in this list aren't tested.
% "newcol" is the name of the column to store category labels in.
%
% "newtable" is a copy of "oldtable" with the category label column added.
```

3.3 nlProc_calcSkewPercentile.m

```
% function [ seriesmedian seriesiqr seriesskew rawpercentiles ] = ...
%   nlProc_calcSkewPercentile(dataseries, tailpercent)
%
% This computes the median and the (tailpercent, 100%-tailpercent) tail
% percentiles for the specified series, and evaluates skew by comparing the
% midsummary (average of the tail values) with the median. The result is
% normalized (a skew of +/- 1 is a displacement by +/- the interquartile
% range).
%
% "dataseries" is the sample sequence to process.
% "tailpercent" is an array of tail values to test.
%
% "seriesmedian" is the series median value.
% "seriesiqr" is the series interquartile range.
```

```
% "seriesskew" is an array of normalized skew values corresponding to the
%   tail percentages.
% "rawpercentiles" is an array with the actual percentile values used for
%   skew calculations. It contains percentile values corresponding to
%   [ (tailpercent) (median) (100 - tailpercent) (25%) (75%) ].
```

3.4 nlProc_calcSpectrumSkew.m

```
% function [ spectfreqs spectmedian spectiqr spectskew ] = ...
%   nlProc_calcSpectrumSkew( dataseries, samprate, ...
%   freqrange, freqperdecade, wintime, winsteps, tailpercent)
%
% This computes a persistence spectrum for the specified series, and finds
% the median, interquartile range, and the normalized skew for each frequency
% bin. "Skew" is defined per nlProc_calcSkewPercentile().
%
% "dataseries" is the data series to process.
% "samprate" is the sampling rate of the data series.
% "freqrange" [ fmin fmax ] specifies the frequency band to evaluate.
% "freqperdecade" is the number of frequency bins per decade.
% "wintime" is the window duration in seconds to compute the time-windowed
%   Fourier transform with.
% "winsteps" is the number of overlapping steps taken when advancing the time
%   window. The window advances by wintime/winsteps seconds per step.
% "tailpercent" is an array of percentile values that define the tails for
%   skew calculation, per nlProc_calcSkewPercentile().
%
% "spectfreqs" is an array of bin center frequencies.
% "spectmedian" is an array of per-frequency median power values.
% "spectiqr" is an array of per-frequency power interquartile ranges.
% "spectskew" is a cell array, with one cell per "tailpercent" value. Each
%   cell contains an array of per-frequency skew values.
```

3.5 nlProc_erodeBooleanCount.m

```
% function newflags = ...
%   nlProc_erodeBooleanCount( oldflags, erodebefore, erodeafter )
%
% This processes a vector of boolean values, eroding "true" flags (extending
% "false" flags) forwards and backwards in time by the specified number of
% samples. Samples up to "erodebefore" at the start of and "erodeafter"
% at the end of sequences of true samples in the original signal are false
% in the returned signal.
%
% Erosion is implemented as dilation of the complement vector with "before"
% and "after" values swapped.
```

```
%
% "oldflags" is the boolean vector to process.
% "erodebefore" is the number of samples at the start of a sequence to squash.
% "erodeafter" is the number of samples at the end of a sequence to squash.
%
% "newflags" is the boolean vector with erosion performed.
```

3.6 nlProc_erodeBooleanTime.m

```
% function newflags = ...
%   nlProc_erodeBooleanTime( oldflags, samprate, erodebefore, erodeafter )
%
% This processes a vector of boolean values, eroding "true" flags (extending
% "false" flags) forwards and backwards in time by the specified durations.
% Samples up to "erodebefore" at the start of and "erodeafter" at the end of
% sequences of true samples in the original signal are false in the returned
% signal.
%
% Erosion is implemented as dilation of the complement vector with "before"
% and "after" values swapped.
%
% "oldflags" is the boolean vector to process.
% "samprate" is the sampling rate of the flag vector.
% "erodebefore" is the duration in seconds at the start of a sequence to
%   squash.
% "erodeafter" is the duration in seconds at the end of a sequence to squash.
%
% "newflags" is the boolean vector with erosion performed.
```

3.7 nlProc_fillNaN.m

```
% function newseries = nlProc_fillNaN( oldseries )
%
% This interpolates NaN segments within the series using linear interpolation,
% and then fills in NaNs at the end of the series by replicating samples.
% This makes the derivative discontinuous when filling endpoints but prevents
% large excursions from curve fit extrapolation.
%
% "oldseries" is the series containing NaN segments.
%
% "newseries" is the interpolated series without NaN segments.
```

3.8 nlProc_filterSignal.m

```
% function [ lfpseries spikeseries ] = nlProc_filterSignal( oldseries, ...
%   samprate, lfprate, lowpassfreq, powerfreq, dcfreq )
%
% This applies several filters:
% - A DC removal filter.
% - A notch filter to remove power line noise.
% - A low-pass filter to isolate the local field potential signal.
% The full-rate LFP series is subtracted from the original series to produce
% a high-pass-filtered "spike" series, and a downsampled LFP series is
% also returned.
%
% "oldseries" is the original wideband signal.
% "samprate" is the sampling rate of the wideband signal.
% "lfprate" is the desired sampling rate of the LFP signal. This should
%   cleanly divide "samprate" (samprate = k * lfprate for some integer k).
% "lowpassfreq" is the edge of the pass-band for the LFP signal. This is
%   lower than the filter's corner frequency; it's the 0.2 dB frequency.
% "powerfreq" is an array of values specifying the center frequencies of the
%   power line notch filter. This is typically 60 Hz or 50 Hz (a single
%   value), but may contain multiple values to filter harmonics. An empty
%   array disables this filter.
% "dcfreq" is the edge of the pass-band for the high-pass DC removal filter.
%   Set to 0 to disable this filter. This is higher than the corner frequency;
%   it's the 0.2 dB frequency (ripple is flat above it).
%
% "lfpseries" is the downsampled low-pass-filtered signal.
% "spikeseries" is the full-rate high-pass-filtered signal.
%
% Filters used are IIR, called with "filtfilt" to remove time offset by
% running the filter forwards and backwards in time. The power line filter
% takes about 1/2 second to fully stabilize, and the low-pass LFP filter takes
% about 1/2 period to 1 period to fully stabilize. Edge effects may occur
% within this distance of the start and end of the signal.
% The DC rejection filter also takes at least 1 period to stabilize. Since
% it's applied in both directions, and won't perturb the pass-band, it should
% be well-behaved over the entire signal.
%
% The LFP sampling rate should be at least 10 times "lowpassfreq" to avoid
% aliasing during downsampling. The DC rejection filter pass frequency
% should be no lower than half the lowest frequency of interest, to
% minimize edge effects.
```

3.9 nlProc_impedanceCalcDistanceToOrthoGauss.m

```
% function sampdistances = nlProc_impedanceCalcDistanceToOrthoGauss( ...
```

```
%   sampmags, sampphases, magmean, magdev, phasemean, phasedev )
%
% Given a set of impedance measurements (or other magnitude/phase data
% points), this computes the distance between each measurement and the
% center of a normal distribution with principal axes parallel to the
% magnitude and phase axes.
%
% Distance is expressed in standard deviations.
%
% "sampmags" is a series of impedance magnitude measurements (typically the
%   logarithm of the actual magnitude).
% "sampphases" is a series of impedance phase measurements, in radians.
% "magmean" is the magnitude distribution's mean.
% "magdev" is the magnitude distribution's standard deviation.
% "phasemean" is the phase distribution's circular mean.
% "phasedev" is the standard deviation of (phase - phase mean). This is
%   assumed to be much smaller than 2pi.
%
% "sampdistances" is a series of scalar values indicating how many standard
%   deviations away from the mean each measurement is. This is the L2 norm
%   of the distances from the magnitude and phase means.
```

3.10 nlProc_impedanceClassifyBox.m

```
% function labels = nlProc_impedanceClassifyBox( ...
%   magdata, phasedata, classdefs, testorder, defaultlabel )
%
% This tests a series of impedance values, applying cluster labels as
% defined by the supplied class definitions. Samples that can't be clustered
% are given a default cluster label.
%
% This function tests against "box" models, as defined in "ZMODELS.txt".
%
% "magdata" is a set of impedance magnitude values. This is typically
%   the logarithm of the actual magnitude.
% "phasedata" is a set of impedance phase values, in radians.
% "classdefs" is a structure indexed by category label, with each field
%   containing a structure that defines the category's cluter, per
%   "ZMODELS.txt". Only "box" definitions are processed by this function.
% "testorder" is a cell array containing category labels, defining the order
%   in which to test for category membership (to disambiguate overlapping
%   categories). The _last_ matching test determines the category label. If
%   this is an empty cell array, an arbitrary order is chosen.
% "defaultlabel" is a character array to be applied as a category label for
%   data points that do not match any cluster definition.
%
% "labels" is a cell array containing cluster labels for each data point.
```

3.11 nlProc_impedanceClassifyOrthoGauss.m

```
% function labels = nlProc_impedanceClassifyOrthoGauss( ...
%   magdata, phasedata, classdefs, maxdistance, defaultlabel )
%
% This tests a series of impedance values, applying cluster labels as
% defined by the supplied class definitions. Samples that can't be clustered
% are given a default cluster label.
%
% This function tests against "orthogauss" models, as defined in
% "ZMODELS.txt".
%
% "magdata" is a set of impedance magnitude values. This is typically
%   the logarithm of the actual magnitude.
% "phasedata" is a set of impedance phase values, in radians.
% "classdefs" is a structure indexed by category label, with each field
%   containing a structure that defines the category's cluter, per
%   "ZMODELS.txt". Only "orthogauss" definitions are processed by this
%   function.
% "maxdistance" is the maximum distance from a cluster center (in standard
%   deviations) that a sample can have while being a member of that cluster.
% "defaultlabel" is a character array to be applied as a category label for
%   data points that do not match any cluster definition.
%
% "labels" is a cell array containing cluster labels for each data point.
```

3.12 nlProc_impedanceFitOrthoGauss.m

```
% function [ magmean, magdev, phasemean, phasedev ] = ...
%   nlProc_impedanceFitOrthoGauss(membermags, memberphases)
%
% Given a set of impedance measurements (or other magnitude/phase data
% points), this independently estimates mean and deviation for impedance
% magnitude and phase angle.
%
% Magnitude uses the arithmetic mean. Phase uses the circular mean, but
% linear deviation (we're assuming deviation is much smaller than 2pi).
%
% "membermags" is a series of magnitude measurements. These are evaluated
%   on a linear scale; they're typically the logarithm of actual magnitude.
% "memberphases" is a series of phase measurements, in radians.
%
% "magmean" is the arithmetic mean of "membermags".
% "magdev" is the standard deviation of "membermags".
% "phasemean" is the circular mean of "memberphases".
% "phasedev" is the standard deviation of (memberphases - phasemean).
```

3.13 nlProc_padBooleanCount.m

```
% function newflags = nlProc_padBooleanCount( oldflags, padbefore, padafter )
%
% This processes a vector of boolean values, extending "true" flags
% forwards and backwards in time by the specified number of samples. Samples
% up to "padbefore" ahead of and "padafter" following true samples in the
% original signal are true in the returned signal.
%
% This is a dilation operation. To perform erosion, perform dilation on the
% complement of a vector (i.e.  newflags = ~ padBooleanCount( ~ oldflags )).
% Remember to swap "before" and "after" for the complement vector.
%
% "oldflags" is the boolean vector to process.
% "padbefore" is the number of samples backwards in time to pad.
% "padafter" is the number of samples forwards in time to pad.
%
% "newflags" is the boolean vector with padding performed.
```

3.14 nlProc_padBooleanTime.m

```
% function newflags = ...
%   nlProc_padBooleanTime( oldflags, samprate, padbefore, padafter )
%
% This processes a vector of boolean values, extending "true" flags
% forwards and backwards in time by the specified durations. Samples
% up to "padbefore" ahead of and "padafter" following true samples in the
% original signal are true in the returned signal.
%
% This is a dilation operation. To perform erosion, perform dilation on the
% complement of a vector (i.e.  newflags = ~ padBooleanTime( ~ oldflags )).
% Remember to swap "before" and "after" for the complement vector.
%
% "oldflags" is the boolean vector to process.
% "samprate" is the sampling rate of the flag vector.
% "padbefore" is the duration in seconds backwards in time to pad.
% "padafter" is the duration in seconds forwards in time to pad.
%
% "newflags" is the boolean vector with padding performed.
```

3.15 nlProc_removeArtifactsSigma.m

```
% function newseries = nlProc_removeArtifactsSigma( oldseries, ...
%   ampthresh, derivthresh, ampthreshfall, derivthreshfall, ...
%   trimbefore, trimafter, smoothsamps, dcsamps )
```

```
%
% This identifies artifacts as excursions in the signal's amplitude or
% derivative, and replaces affected regions with NaN. Excursion thresholds
% are expressed in terms of the standard deviation of the signal or its
% derivative.
%
% "oldseries" is the series to process.
% "ampthresh" is the threshold for flagging amplitude excursion artifacts.
% "derivthresh" is the threshold for flagging derivative excursion artifacts.
% "ampthreshfall" is the turn-off threshold for amplitude artifacts.
% "derivthreshfall" is the turn-off threshold for derivative artifacts.
% "trimbefore" is the number of samples to squash ahead of the artifact.
% "trimafter" is the number of samples to squash after the artifact.
% "smoothsamps" is the size of the smoothing window to apply before taking
%   the derivative, or 0 for no smoothing.
% "dcsamps" is the size of the window for computing local DC average removal
%   ahead of computing signal statistics.
%
% Regions where the amplitude or derivative exceeds the threshold are flagged
% as artifacts. These regions are widened to encompass the region where the
% amplitude or derivative exceeds the "fall" threshold, and then padded by
% the specified number of samples. This is intended to correctly handle
% square-pulse artifacts and fast-step-slow-decay artifacts.
```

3.16 nlProc_removeTimeRanges.m

```
% function newseries = ...
%   nlProc_removeTimeRanges( oldseries, samprate, trimtimes )
%
% This NaNs out specified regions of the input signal.
%
% "oldseries" is the series to process.
% "samprate" is the sampling rate of the input signal.
% "trimtimes" is a cell array containing time spans to NaN out. Time spans
%   have the form "[ time1 time2 ]", where times are in seconds. Negative
%   times are relative to the end of the signal, positive times are relative
%   to the start of the signal (both start at 0 seconds). Use a very small
%   negative value for "-0".
%
% "newseries" is a modified version of the input series with the specified
%   time ranges set to NaN.
```

3.17 nlProc_rollAndPadCount.m

```
% function newseries = ...
%   nlProc_rollAndPadCount( oldseries, rollsamps, padsamps )
```

```
%
% This performs DC and ramp removal, applies a Tukey (cosine) roll-off
% window, and pads the endpoints of the supplied signal.
%
% "oldseries" is the series to process.
% "rollsamps" is the length in samples of the starting and ending roll-offs.
% "padsamps" is the number of starting and ending padding samples to add.
%
% "newseries" is the processed signal.
```

3.18 nlProc_rollAndPadTime.m

```
% function newseries = ...
%   nlProc_rollAndPadTime( oldseries, samprate, rolltime, padtime )
%
% This performs DC and ramp removal, applies a Tukey (cosine) roll-off
% window, and pads the endpoints of the supplied signal.
%
% "oldseries" is the series to process.
% "samprate" is the sampling rate of the input signal.
% "rolltime" is the duration in seconds of the starting and ending roll-offs.
% "padtime" is the duration in seconds of starting and ending padding.
%
% "newseries" is the processed signal.
```

3.19 nlProc_trimEndpointsCount.m

```
% function newseries = ...
%  nlProc_trimEndpointsCount( oldseries, trimstart, trimend )
%
% This crops the specified number of samples from the start and end of the
% supplied signal.
%
% "oldseries" is the series to process.
% "trimstart" is the number of samples to remove from the beginning.
% "trimend" is the number of samples to remove from the end.
%
% "newseries" is a truncated version of the input signal.
```

3.20 nlProc_trimEndpointsTime.m

```
% function newseries = ...
%   nlProc_trimEndpointsTime( oldseries, samprate, trimstart, trimend )
```

```
%
% This crops the specified durations from the start and end of the supplied
% signal.
%
% "oldseries" is the series to process.
% "samprate" is the sampling rate of the input signal.
% "trimstart" is the number of seconds to remove from the beginning.
% "trimend" is the number of seconds to remove from the end.
%
% "newseries" is a truncated version of the input signal.
```

Chapter 4

"nlUtil" Functions

4.1 nlUtil_forceColumn.m

```
% function newseries = nlUtil_forceColumn(oldseries)
%
% This function forces a one-dimensional vector into Nx1 form.
%
% "oldseries" is a 1xN or Nx1 vector.
%
% "newseries" is the corresponding Nx1 vector.
```

4.2 nlUtil_forceRow.m

```
% function newseries = nlUtil_forceRow(oldseries)
%
% This function forces a one-dimensional vector into 1xN form.
%
% "oldseries" is a 1xN or Nx1 vector.
%
% "newseries" is the corresponding 1xN vector.
```

4.3 nlUtil_structToTable.m

```
% function outdata = nlUtil_structToTable(indata, ignorelist)
%
% This converts a structure into a table. Table columns correspond to
% structure fields. Structure vectors forced to Nx1 (column) format.
% Specified structure fields may be skipped.
% NOTE - Data fields must all have the same length!
```

```
%
% "indata" is the structure to convert.
% "ignorelist" is a cell array containing structure field names to skip.
%
% "outdata" is a table with columns containing structure field data.
```

4.4 nlUtil_tableToStruct.m

```
% function outdata = nlUtil_tableToStruct(indata, rowcol)
%
% This converts a table into a structure. Structure fields correspond to
% table columns, and are stored as either 1xN or Nx1 vectors.
%
% "indata" is the table to convert.
% "rowcol" is 'row' to output 1xN vectors and 'col' for Nx1 vectors.
%
% "outdata" is a structure containing table column data.
```

Chapter 5

"nlPlot" Functions

5.1 nlPlot_axesPlotExcursions.m

```
% function nlPlot_axesPlotExcursions( thisax, ...
%   spectfreqs, spectmedian, spectiqr, spectskew, percentlist, ...
%   want_relative, figtitle )
%
% This plots LFP power excursions, either as relative power excess alone or
% against the median power spectrum. See nlChan_applySpectSkewCalc() for
% details of skew calculation and array contents.
% The plot is rendered to the specifed set of figure axes.
%
% "thisax" is the "axes" object to render to.
% "spectfreqs" is an array of frequency bin center frequencies.
% "spectmedian" is an array of per-frequency median power values.
% "spectiqr" is an array of per-frequency power interquartile ranges.
% "spectskew" is a cell array, with one cell per "percentlist" value. Each
%   cell contains an array of per-frequency skew values.
% "percentlist" is an array of percentile values that define the tails for
%   skew calculations, per nlProc_calcSkewPercentile().
% "want_relative" is true to plot relative power excess alone, and false to
%   plot against the median power spectrum.
% "figtitle" is the title to use for the figure, or '' for no title.
```

5.2 nlPlot_axesPlotPersist.m

```
% function nlPlot_plotPersist( thisax, ...
%   persistvals, persistfreqs, persistpowers, want_log, figtitle )
%
% This plots a pre-tabulated persistence spectrum. See "pspectrum()" for
% details of input array structure.
%
```

```
% "thisax" is the "axes" object to render to.
% "thisfig" is the figure to render to (this may be a UI component).
% "persistvals" is the matrix of persistence spectrum fraction values.
% "persistfreqs" is the list of frequencies used for binning.
% "persistpowers" is the list of power magnitudes used for binning.
% "want_log" is true if the frequency axis should be plotted on a log scale
%   (it's computed on a linear scale).
% "figtitle" is the title to use for the figure, or '' for no title.
```

5.3 nlPlot_axesPlotSpikeHist.m

```
% function nlPlot_plotSpikeHist( thisax, ...
%   bincounts, binedges, percentamps, percentpers, figtitle )
%
% This plots a pre-tabulated histogram of normalized spike waveform
% amplitude. For channels with real spikes, tails are asymmetrical.
%
% "thisax" is the "axes" object to render to.
% "bincounts" is an array containing bin count values, per histogram().
% "binedges" is an array containing the histogram bin edges, per histogram().
% "percentamps" is an array of normalized amplitudes corresponding to desired
%   tail percentiles to highlight. Entries 1..N are tail percentile amplitudes,
%   entry N+1 is the median, and entries N+2..2N+1 are (100%-tail) amplitudes.
% "percentpers" is an array naming desired tail percentiles to highlight.
% "figtitle" is the title to use for the figure, or '' for no title.
```

5.4 nlPlot_getColorLUTPeriodic.m

```
% function collut = nlPlot_getColorLUTPeriodic()
%
% This function returns a cell array of color triplets. Colors are chosen
% so that successive colors are similar and so that the list may be iterated
% through repeatedly.
%
% "collut" is a cell array containing color triplets.
```

5.5 nlPlot_getColorPalette.m

```
% function cols = nlPlot_getColorPalette()
%
% This function returns a structure containing color triplets indexed by
% color name.
%
```

```
% Colors suppled are "blu", "brn", "yel", "mag", "grn", "cyn", and "red".
% These are mostly cribbed from get(cga,'colororder'), with tweaks.
%
% "cols" is a structure containing color triplets.
```

5.6 nlPlot_getColorSpread.m

```
% function colorlist = nlPlot_getColorSpread(origcol, count, anglespan)
%
% This function takes a starting color and turns it into a spectrum of
% nearby colors, by walking around the color wheel starting with the original
% color.
%
% The resulting sequence is returned as a cell array of color vectors.
%
% "origcol" [ r g b ] is the starting color.
% "count" is the number of colors to return.
% "anglespan" (degrees) is the distance to walk along the color wheel.
%
% "colorlist" is a cell array of the resulting [r g b] color vectors.
```

5.7 nlPlot_plotExcursions.m

```
% function nlPlot_plotExcursions( thisfig, oname, ...
%   spectfreqs, spectmedian, spectiqr, spectskew, percentlist, ...
%   want_relative, figtitle )
%
% This plots LFP power excursions, either as relative power excess alone or
% against the median power spectrum. See nlChan_applySpectSkewCalc() for
% details of skew calculation and array contents.
%
% "thisfig" is the figure to render to (this may be a UI component).
% "oname" is the filename to save to, or '' to not save.
% "spectfreqs" is an array of frequency bin center frequencies.
% "spectmedian" is an array of per-frequency median power values.
% "spectiqr" is an array of per-frequency power interquartile ranges.
% "spectskew" is a cell array, with one cell per "percentlist" value. Each
%   cell contains an array of per-frequency skew values.
% "percentlist" is an array of percentile values that define the tails for
%   skew calculations, per nlProc_calcSkewPercentile().
% "want_relative" is true to plot relative power excess alone, and false to
%   plot against the median power spectrum.
% "figtitle" is the title to use for the figure, or '' for no title.
```

5.8 nlPlot_plotPersist.m

```
% function nlPlot_plotPersist( thisfig, oname, ...
%   persistvals, persistfreqs, persistpowers, want_log, figtitle )
%
% This plots a pre-tabulated persistence spectrum. See "pspectrum()" for
% details of input array structure.
%
% "thisfig" is the figure to render to (this may be a UI component).
% "oname" is the filename to save to, or '' to not save.
% "persistvals" is the matrix of persistence spectrum fraction values.
% "persistfreqs" is the list of frequencies used for binning.
% "persistpowers" is the list of power magnitudes used for binning.
% "want_log" is true if the frequency axis should be plotted on a log scale
%   (it's computed on a linear scale).
% "figtitle" is the title to use for the figure, or '' for no title.
```

5.9 nlPlot_plotSpikeHist.m

```
% function nlPlot_plotSpikeHist( thisfig, oname, ...
%   bincounts, binedges, percentamps, percentpers, figtitle )
%
% This plots a pre-tabulated histogram of normalized spike waveform
% amplitude. For channels with real spikes, tails are asymmetrical.
%
% "thisfig" is the figure to render to (this may be a UI component).
% "oname" is the filename to save to, or '' to not save.
% "bincounts" is an array containing bin count values, per histogram().
% "binedges" is an array containing the histogram bin edges, per histogram().
% "percentamps" is an array of normalized amplitudes corresponding to desired
%   tail percentiles to highlight. Entries 1..N are tail percentile amplitudes,
%   entry N+1 is the median, and entries N+2..2N+1 are (100%-tail) amplitudes.
% "percentpers" is an array naming desired tail percentiles to highlight.
% "figtitle" is the title to use for the figure, or '' for no title.
```

Chapter 6

"nlIO" Functions

6.1 nlIO_formatMemberList.m

```
% function memberstring = ...
%   nlIO_formatMemberList( candidates, format, memberflags )
%
% This formats a list of matching candidates in human-readable form, in a
% manner similar to page numbering (e.g. "5-6, 7, 12, 13-15"). Member
% entries that are contiguous in the candidate list are reported as ranges
% rather than as individuals.
%
% The candidate list is assumed to already be sorted in a sensible order.
%
% "candidates" is a vector or cell array containing member labels.
% "format" is a "sprintf" conversion format for turning a candidate label
%   into appropriate human-readable output.
% "memberflags" is a logical vector of the same size as "candidates" that is
%   "true" for candidates that are to be reported and "false" otherwise.
%
% "memberstring" is a human-readable string summarizing the list of
%   candidates for which "memberflags" is true.
```

6.2 nlIO_formatTableData.m

```
% function newtable = nlIO_formatTableData( oldtable, formatlut )
%
% This function converts the specified data columns in oldtable into strings,
% using "sprintf" with the format specified for that column's entry in the
% lookup table.
%
% "oldtable" is the table to convert.
% "formatlut" is a "containers.Map" object mapping table column names to
```

```
%   sprintf format character arrays.
%
% "newtable" is a copy of "oldtable" with the specified columns converted.
```

6.3 nlIO_quoteTableStrings.m

```
% function newtable = nlIO_quoteTableStrings(oldtable)
%
% This function returns a copy of the input table with all string cell values
% and all column names in quotes.
%
% This is a workaround for an issue with "writetable". If "writetable" is
% called to write CSV with 'QuoteStrings' set to true, only cell values are
% quoted, not column names. If column names are manually quoted, they get
% triple quotes. The solution is to set 'QuoteStrings' false and manually
% quote all strings and all column names, which this function does.
```

6.4 nlIO_readAndBinImpedance.m

```
% function ztable = nlIO_readAndBinImpedance( fnamelist, ...
%   chancolumn, magcolumn, phasecolumn, phaseunits, bindefs, testorder )
%
% This reads one or more CSV files containing channel impedance measurements.
% Impedance measurements are averaged across files, and channels are tagged
% with type labels based on user-specified criteria (typical types are
% high-impedance, low-impedance, grounded, and floating).
%
% For automated clustering, supply an empty cell array for "testorder"
% (the contents of "bindefs" are ignored in this situation).
%
% When multiple measurements for a given channel ID label are present,
% the magnitude is averaged using the geometric mean (to tolerate large
% differences in magnitude) and the phase angle is averaged using
% circular statistics (mean direction).
%
% NOTE - Phase units are not modified, but we need to know what the units
% are in order to compute the mean direction when averaging phase samples.
%
% NOTE - Phase is wrapped to +/- 180 deg (+/- pi radians).
%
% NOTE - This needs Matlab R2019b or later for 'PreserveVariableNames'.
% It'll still work in older versions but will alter column names to be
% Matlab-safe (use matlab.lang.makeValidName() to duplicate this).
%
% "fnamelist" is a cell array containing the names of files to read. These
%   are expected to be CSV files.
```

```
% "chancolumn" is the table column to read channel ID labels from.
% "magcolumn" is the table column to read impedance magnitude from.
% "phasecolumn" is the table column to read impedance phase from.
% "phaseunits" is 'degrees' or 'radians'.
% "bindefs" is a category definition structure per "nlProc_binTableDataSimple".
% "testorder" is the order in which to test category definitions. The first
%   label is the default label, and the _last_ label with a successful test
%   is applied. If this is empty, it forces automatic cluster detection.
%
% "ztable" is a table containing the following columns:
%   "label" is a copy of the "chancolumn" input column.
%   "magnitude" is a copy of the "magcolumn" input column.
%   "phase" is a copy of the "phasecolumn" input column.
%   "type" is the category label for each channel.
```

6.5 nlIO_readBinaryFile.m

```
% function [is_ok sampdata] = nlIO_readBinaryFile(fname, dtype)
%
% This attempts to read a packed array of the specified data type from the
% specified file.
%
% "fname" is the name of the file to read from.
% "dtype" is a string identifying the Matlab data type (e.g. 'uint32').
%
% "is_ok" is set to true if the operation succeeds and false otherwise.
% "sampdata" is an array containing the sample values.
```

6.6 nlIO_writeBinaryFile.m

```
% function is_ok sampdata = nlIO_writeBinaryFile(fname, sampdata, dtype)
%
% This attempts to write the specified sample data as a packed array of the
% specified data type. Per fwrite(), data is rounded and saturated if
% appropriate.
%
% "fname" is the name of the file to write to.
% "sampdata" is an array containing the sample values.
% "dtype" is a string identifying the Matlab data type (e.g. 'uint32').
%
% "is_ok" is set to true if the operation succeeds and false otherwise.
```

Chapter 7

"nlIntan" Functions

7.1 nlIntan_getAmpChannelFilename.m

```
% function fname = nlIntan_getAmpChannelFilename(indir, bankid, channum)
%
% This returns the data file name for the specified Intan amplifier channel.
% This file doesn't necessarily exist; this just constructs the name.
%
% "indir" is the directory containing Intan data.
% "bankid" is the bank label for the desired channel.
% "channum" is the in-bank channel number for the desired channel.
%
% "fname" is the name of the file that should contain the channel's data.
% This is an empty character array if an error occurred.
```

7.2 nlIntan_getTimeFilename.m

```
% function fname = nlIntan_getTimeFilename(indir)
%
% This returns the name of the file containing Intan signal timestamps.
% The file doesn't necessarily exist; this just constructs the filename.
% NOTE - Intan saves the sample indices, not an actual time values.
%
% "indir" is the directory to search.
%
% "fname" is the name of the file that should contain sample time indices.
```

7.3 nlIntan_helper_probeChannels.m

```
% function [ chandetect chanfiles ] = ...
%   nlIntan_helper_probeChannels(chantest, banklist, chanrange, fnamefunc)
%
% This checks for the existence of data files from specified I/O banks, and
% and probes for channels and banks if asked to do so.
%
% "chantest" is a structure with field names that are bank identifiers
%   ('A', 'B', 'DIN', etc), with each field containing an array of channel
%   numbers to test. See "CHANLIST.txt" for details.
%   An empty structure means "auto-detect all banks". An empty channel array
%   means "auto-detect all channels for this bank".
% "bankids" is a cell array containing a list of bank IDs that are
%   potentially probed.
% "chanrange" is an array of channel numbers that are potentially probed.
% "fnamefunc" points to a function that constructs a filename when passed
%   a bank ID and a channel number as arguments.
%
% "chandetect" is a structure with the same format as "chantest", enumerating
%   the banks and channels from which data was read. Fields in "chantest"
%   that are not "potentially probed" bank identifiers are copied as-is.
% "chanfiles" is an array of structures containing the following fields:
%    "bank"  - Bank identifier (field name).
%    "chan"  - Channel number.
%    "fname" - Name of the file containing this bank/channel's data.
```

7.4 nlIntan_probeAmpChannels.m

```
% function [ chandetect ampdetect chanfiles ] = ...
%   nlIntan_probeAmpChannels(indir, chantest)
%
% This checks for the existence of data files from specified amplifier
% channels, and probes for channels and amplifiers if asked to do so.
%
% "indir" is the directory to search.
% "chantest" is a structure with field names that are amplifier identifiers
%   ('A', 'B', etc), with each field containing an array of channel numbers
%   to fetch. An empty structure means "auto-detect all amplifiers". An
%   empty array in a channel field means "auto-detect all channels".
%
% "chandetect" is a structure with the same format as "chantest", enumerating
%   the amplifiers and channels from which data was read. Fields in "chantest"
%   that are not amplifier identifiers (per CHANLIST.txt) are copied as-is.
% "ampdetect" is a cell array of field names, containing only detected
%   amplifier IDs.
% "chanfiles" is an array of structures containing the following fields:
```

```
%   "bank"  - Amplifier identifier string.
%   "chan"  - Channel number.
%   "fname" - Name of the file containing this bank/channel's data.
```

7.5 nlIntan_readAmpChannels.m

```
% function [is_ok chandetect ampdetect timedata chandata] = ...
%   nlIntan_readAmpChannels(indir, chanlist)
%
% This attempts to read individual Intan amplifier channel data files from
% the specified directory. The time series is also read.
%
% "indir" is the directory to search.
% "chanlist" is a structure with field names that are amplifier identifiers
%   ('A', 'B', etc), with each field containing an array of channel numbers
%   to fetch. An empty structure means "auto-detect all amplifiers". An
%   empty array in a channel field means "auto-detect all channels".
%
% "is_ok" is true if data was read and false otherwise.
% "chandetect" is a structure with the same format as "chanlist", enumerating
%   the amplifiers and channels from which data was read.
% "timedata" contains the time series (in samples, not seconds).
% "chandata" is an array of structures containing the following fields:
%   "bank" - Amplifier identifier string.
%   "chan" - Channel number.
%   "fname" - Filename.
%   "data" - Sample data series.
```

7.6 nlIntan_readMetadata.m

```
% function [ is_ok metadata ] = nlIntan_readMetadata(fname)
%
% This attempts to read selected parts of the specified Intan metadata file.
% If successful, "is_ok" is set to "true" and "metadata" is a structure with
% the following fields:
%
% "devtype"  - "RHS" for a recording controller, "RHD" for stimulation.
% "samprate" - Sampling rate in Hz.
%
% On failure, "is_ok" is set to "false" and "metadata" is an empty structure.
%
% FIXME - Ignoring most of the metadata. Use Intan's functions if you need
% all of it.
```

Chapter 8

"nlChan" Functions

8.1 nlChan_applyArtifactReject.m

```
% function [ newdata fracbad ] = nlChan_applyArtifactReject( ...
%   wavedata, samprate, tuningparams, keepnan )
%
% This performs truncation and artifact rejection, optionally followed by
% interpolation in the former artifact regions.
%
% "wavedata" is the waveform to process.
% "refdata" is a reference to subtract from the waveform, or [] for no
%   reference. The reference should already be truncated and have artifacts
%   removed, but should retain NaN values to avoid introducing new artifacts.
% "samprate" is the sampling rate.
% "tuningparams" is a structure containing tuning parameters for artifact
%   rejection.
% "keepnan" is true if NaN values are to remain and false if interpolation
%   is to be performed to remove them.
%
% "newdata" is the series after artifact removal.
% "fracbad" is the fraction of samples discarded as artifacts (0..1).
```

8.2 nlChan_applyFiltering.m

```
% function [ lfpseries spikeseries ] = nlChan_applyFiltering( ...
%   wavedata, samprate, tuningparams );
%
% This performs filtering to suppress power line noise, zero-average the
% signal, and to split the signal into LFP and spike components.
%
% Power line noise filtering and DC removal filtering can be suppressed by
% setting their respective filter frequencies to 0 Hz.
```

```
%
% "wavedata" is the waveform to process.
% "samprate" is the sampling rate.
% "tuningparams" is a structure containing tuning parameters for filtering.
%
% "newdata" is the series after filtering.
```

8.3 nlChan_applySpectSkewCalc.m

```
% function [ spectfreqs spectmedian spectiqr spectskew ] = ...
%   nlChan_applySpectSkewCalc( wavedata, samprate, tuningparams, perclist )
%
% This calls nlProc_calcSpectrumSkew() to compute a persistence spectrum for
% the specified series and to compute statistics and skew for each frequency
% bin.
%
% "wavedata" is the waveform to process.
% "samprate" is the sampling rate.
% "tuningparams" is a structure containing tuning parameters for persistence
%   spectrum generation.
% "perclist" is an array of percentile values that define the tails for
%   skew calculation, per nlProc_calcSkewPercentile().
%
% "spectfreqs" is an array of bin center frequencies.
% "spectmedian" is an array of per-frequency median power values.
% "spectiqr" is an array of per-frequency power interquartile ranges.
% "spectskew" is a cell array, with one cell per "perclist" value. Each cell
%   contains an array of per-frequency skew values.
```

8.4 nlChan_getArtifactDefaults.m

```
% function paramstruct = nlChan_getArtifactDefaults()
%
% This returns a structure containing reasonable default tuning parameters
% for artifact rejection.
%
% Parameters that will most often be varied are "ampthresh", "diffthresh",
% "trimstart", and "trimend".
```

8.5 nlChan_getFilterDefaults.m

```
% function paramstruct = nlChan_getFilterDefaults()
%
```

```
% This returns a structure containing reasonable default tuning parameters
% for signal filtering.
%
% Parameters that will most often be varied are "powerfreq" and "lfprate".
```

8.6 nlChan_getPercentDefaults.m

```
% function paramstruct = nlChan_getPercentDefaults()
%
% This returns a structure containing reasonable default tuning parameters
% for spike and burst identification via percentile binning.
%
% Parameters that will most often be varied are "burstselectidx" and
% "spikeselectidx".
```

8.7 nlChan_getSpectrumDefaults.m

```
% function paramstruct = nlChan_getSpectrumDefaults()
%
% This returns a structure containing reasonable default tuning parameters
% for persistence spectrum generation.
```

8.8 nlChan_iterateChannels.m

```
% function outdata = ...
%   nlChan_iterateChannels(chanfiles, bankrefs, samprate, refparams, procfunc)
%
% This iterates through a list of channel records, loading and preprocessing
% each channel and then calling a processing function with the channel
% data. Processing output is aggregated and returned.
%
% "chanfiles" is an array of channel file records in the format returned by
%   nlIntan_probeAmpChannels(). These have the following fields:
%   "bank" - Amplifier identifier string.
%   "chan" - Channel number.
%   "fname" - Name of the file containing this bank/channel's data.
% "bankrefs" is a structure with field names that are bank identifiers, with
%   each field containing a single channel number specifying the in-bank
%   channel to use as a reference for the remaining channels. If an empty
%   array is present or if a bank identifier is absent, no reference is used
%   for that bank. These channels must also be present in "chanfiles".
% "samprate" is the sampling rate.
% "tuningart" is a structure containing tuning parameters for artifact
```

```
%   rejection.
% "procfunc" is a function handle that is called per file. It has the form:
%     resultval = procfunc(chanrec, wavedata)
%   This is typically an anonymous function that wraps a function with
%   additional arguments.
%
% "outdata" is a copy of "chanfiles" augmented with an additional "result"
%   field, containing "resultval" returned from procfunc(). Only records
%   corresponding to channels that were processed are present; channels that
%   were discarded due to artifacts or that were references are absent.
```

8.9 nlChan_processChannel.m

```
% function resultstats = nlChan_processChannel( wavedata, samprate, ...
%   tuningfilt, tuningspect, tuningperc )
%
% This accepts a wideband waveform, performs filtering to split it into
% spike and LFP signals, and calculates various statistics for each of these
% signals.
%
% This is intended to be called by nlChan_iterateChannels() via a wrapper.
%
% "wavedata" is the waveform to process.
% "samprate" is the sampling rate.
% "tuningfilt" is a structure containing tuning parameters for filtering.
% "tuningspect" is a structure containing tuning parameters for persistence
%   spectrum generation.
% "tuningperc" is a structure containing tuning parameters for spike and
%   burst identification via percentile binning.
%
% "resultstats" is a structure containing the following fields:
%   "spikemedian", "spikeiqr", "spikeskew", and "spikepercentvals" are the
%     corresponding fields returned by nlProc_calcSkewPercentile() using the
%     high-pass-filtered spike signal.
%   "spikebincounts" and "spikebinedges" are the the corresponding fields
%     returned by histcounts() using a normalized version of the spike signal.
%   "spectfreqs", "spectmedian", "spectiqr", and "spectskew" are the
%     corresponding fields returned by nlChan_applySpectSkewCalc() using the
%     low-pass-filtered LFP signal.
%   "persistvals", "persistfreqs", and "persistpowers" are the corresponding
%     fields returned by pspectrum() using the LFP signal.
```

8.10 nlChan_rankChannels.m

```
% function [ bestlist typbest typmiddle typworst ] = ...
%   nlChan_rankChannels( chanrecs, maxperbank, typfrac, scorefunc )
```

```
%
% This process a list of channel records returned by nlChan_iterateChannels().
% Channel records receive a score, and the list is sorted by that score.
% Statistics for typical records and aggregate statistics are returned.
% The sorted list is pruned to include at most a certain number of channels
% per bank, and is then returned.
% NOTE - The "typical" records are not necessarily in the pruned result list.
%
% "chanrecs" is the list of channel statistics records to process.
% "maxperbank" is the maximum number of channels per bank in the returned list.
% "typfrac" is the percentile for finding "typical" good and bad records.
%   This is a number between 0 and 50 (typically 5, 10, or 25).
% "scorefunc" is a function handle that is called for each channel. It has
%   the form:
%     scoreval = scorefunc(resultval)
%   The "resultval" argument is the chanrecs(n).result field, per
%   nlChan_iterateChannels().
%   Higher scores are better, for purposes of this function. A score of NaN
%   squashes a result (removing it from the result list).
%
% "bestlist" is a subset of the sorted channel record list containing the
%   highest-scoring entries subject to the constraints described above.
% "typbest" is the channel record for the top Nth percentile channel.
% "typmiddle" is the channel record for the median channel.
% "typworst" is the channel record for the bottom Nth percentile channel.
```