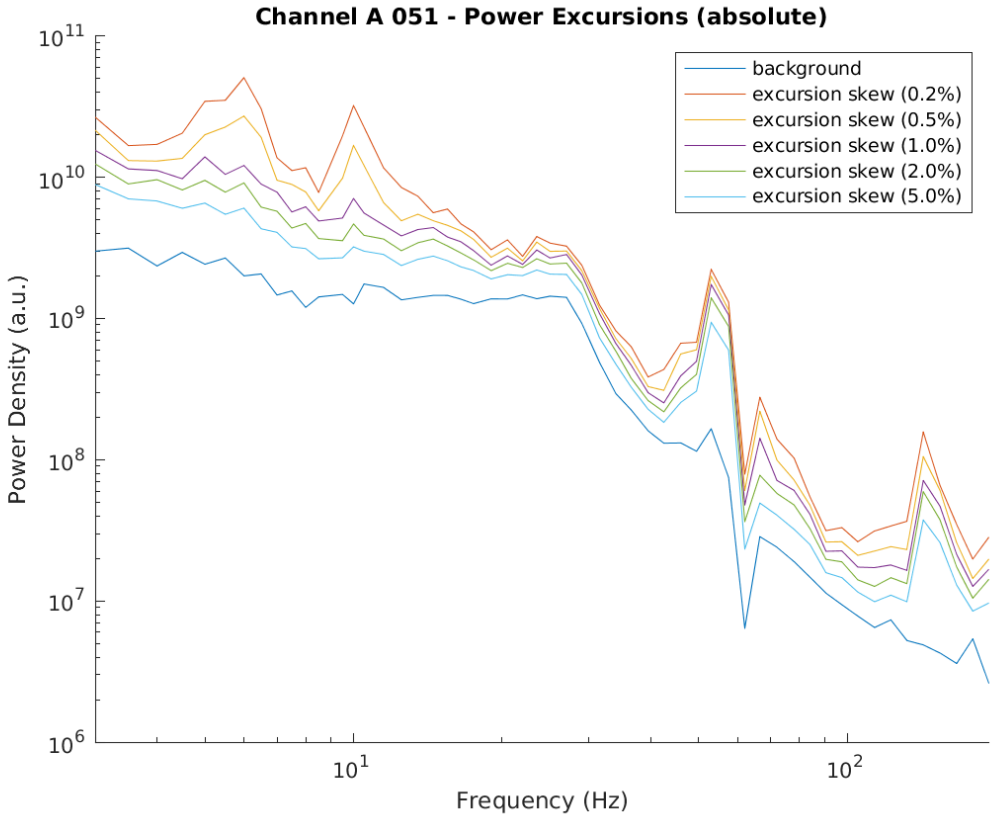
NeuroLoop Utilities – Function Reference

Written by Christopher Thomas – January 8, 2024.



Overview

The NeuroLoop utility library functions are divided into several categories:

- **"Core"** functions (part I) are functions useful in a wide variety of contexts (not vendor- or application-specific).
- **"Abstraction"** functions (part II) are functions used to support specific hardware devices or software suites.
- **"Application"** functions (part III) are functions used by individual application programs that are either designed specifically to interact with that application's code or that perform tasks specific enough to limit reusability.
- Lastly, part IV contains sample code that may help illustrate the ways these libraries may be used.

"Core" function libraries are as follows:

- **"Artifact"** functions (ch. 2) perform artifact rejection via various methods.
- **"Basis"** functions (ch. 3) use various methods to express several channels' signals as the sum of a small number of basis signals. Among other things this is useful for background subtraction.
- **"I/O"** functions (ch. 4) facilitate operations for reading and writing data that aren't vendor-specific.
- **"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.
- **"Processing"** functions (ch. 6) perform signal processing operations on data series such as filtering, artifact rejection, statistical calculations, and so forth.
- **"Synth"** functions (ch. 7) generate synthetic signals using various models.
- **"Utility"** functions (ch. 8) perform miscellaneous operations that are not covered by the other categories.

"Abstraction" function libraries are as follows:

- **"Field Trip"** functions (ch. 10) facilitate interoperation with Field Trip.
- **"Intan"** functions (ch. 11) facilitate the use of datasets stored in Intan's format.
- **"Vendor-Supplied Intan"** functions (ch. 12) are functions derived from vendor-supplied code that are in turn wrapped by the functions in ch. 11.
- **"Open Ephys"** functions (ch. 13) facilitate the use of datasets stored in Open Ephys's format.

"Application" library functions are as follows:

- **"Channel Tool"** functions (ch. 15) 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.
- **"Sanity check"** functions (ch. 16) perform operations used by the in-house dataset "sanity checking" utility. These functions encapsulate operations that are not lab-specific.

Relevant notes about data structures and file structures used by the various libraries are described in their associated "Additional Notes" chapters.

Contents

Part I

Core Libraries

Chapter 1

Core Library Structures and Additional Notes

1.1 ARTFITPARAMS.txt

```
An artifact curve fit parameter structure describes a curve fit performed
by one of the nlArt_fitXX functions.


Fields common to all types:

"fittype" is a character vector indicating the curve fit model used.
  'exp' is a first-order exponential decay fit.



First-order exponential decay fit:

This has the form:   f(t) = coeff * exp(t/tau) + offset

"fittype" is 'exp'.
Parameter fields are:  "coeff", "tau", "offset"



(This is the end of the file.)
```

1.2 BASISVECTORS.txt

```
The "nlBasis_getBasisVectors_XX" functions decompose a set of input signals
into weighted sums of a set of basis vectors, plus a common background.
```

Input signals are stored as is a Nvectors x Ntimesamples matrix of sample vectors. For ephys data, this is typically Nchans x Ntimesamples.

A basis decomposition is stored as a structure with the following fields:

"basisvecs" is a Nbasis x Ntimesamples matrix where each row is a basis vector.
"coeffs" is a Nvectors x Nbasis matrix with basis vector weights for each input vector.
"background" is a 1 x Ntimesamples vector containing a constant background to be added to all vectors during reconstruction. This is typically, but not always, either zero or the mean across sample vectors.

The input is reconstructed as:

```
recon = coeffs * basisvecs + repmat( background, Nvectors, 1 );
```

(This is the end of the file.)

1.3 CHANLIST.txt

Channel lists are used for several purposes. They may be used to specify a set of channels, or they may be used to store per-channel information such as reference selection or physical source channel in a channel map.

A "project-level channel list" is a structure with one field per folder, indexed by user-assigned folder label. Each folder's field contains a "folder-level channel list".

A "folder-level channel list" is a structure with one field per bank, indexed by bank label. Each bank's field contains a "bank-level channel list".

A "bank-level channel list" is a structure with the following fields:

- "chanlist" is a vector containing integer-valued channel indices.
- "scalarmeta" (optional) is a structure containing metadata that isn't stored per-channel.

```
Other optional fields are vectors or cell arrays with per-channel metadata.

Situation-specific optional fields are described below:

For channel lists specifying data to be read:
- "samprange" (optional) [firstsamp lastsamp] is a two-element vector
  specifying the range of samples to read. Sample index 1 is the first sample.
  If "samprange" isn't given or is empty ([]), all samples are read.

For channel lists derived from metadata:
- "scalarmeta.banktype" is a copy of the signal bank metadata's "banktype"
  field.

For reference selection:
- "reflist" is a cell array containing user-specified labels of references
  to use for each channel, when re-referencing. A label that's an empty
  character array means "don't re-reference".
  (References are typically defined as channel lists stored elsewhere.)

For channel mapping:
- "foldersrc" is a cell array containing the folder labels of the source
  channels for each channel in "chanlist".
- "banksrc" is a cell array containing the bank labels of the source
  channels for each channel in "chanlist".
- "chansrc" is a vector containing channel indices of the source channels
  for each channel in "chanlist".

For per-channel signal processing:
- "resultlist" is a cell array containing the output of the signal processing
  function that was called for each channel (per PROCFUNC.txt).



This is the end of the file.
```

1.4 EXPGUESSCONFIG.txt

```
Exponential fits with guessed locations are configured using a structure with
the following fields:

"max_curves" is the number of components to curve fit.

"fitmethod" is a character vector specifying the curve fit algorithm used.
  This is 'log' or 'pinmax', per nlArt_fitExpDecay().

"want_pin_minimum" is true to perform one additional 'pinmax' curve fit at
  the "min_fit_offset_ms" time, and false otherwise.
```

"lowpass_corner" is the corner frequency in Hz of the low-pass filter
  to apply before performing curve fits, or NaN to not filter.

"lowpass_squash_ms" [start stop] is a time range to squash before
  performing low-pass filtering, in milliseconds. This is intended to
  suppress ringing from artifacts. Set this to [] to not squash.

"full_time_range_ms" is the time range over which curve fitting is to be
  performed, in milliseconds.

"min_fit_offset_ms" is the minimum offset relative to the start of
  full_time_range_ms at which fits are to be performed. The intent is to
  avoid ill-behaved regions at the exact time of stimulation.

"post_level_span" [min max] is the fraction of the time range to use for
  estimating the after-setting DC level (e.g. [0.5 0.9]).

"from_detect_level_span" [min max] is the time range to use for updating
  the DC level estimate after curve fitting, as a multiple of the
  excursion detection time within the full time range (e.g. [1.0 2.0]).

"detect_threshold" is the amount by which the signal must depart from the
  DC level for an artifact to be considered present. This is a multiple
  of the median-to-quartile distance (about two thirds of a standard
  deviation). Typical values are 6-12 for clear exponential excursions.

"fit_range" [min max] is a multiplier to apply to the excursion detection
  time to get the time range over which to curve fit (e.g. [0.7 3.0]).

"next_mult" is a multiplier to apply to the excursion detection time to get
  the maximum acceptable detection time for the next excursion (e.g. 0.9).
  This is to prevent the algorithm from continually re-fitting the same
  span (which it can do if it's having trouble getting a good curve fit).

"detect_max_start" is a relative position within the full time range
  (e.g. 0.3). Excursions detected after this time are assumed to be
  spurious (wandering DC level).

"min_detect_ms" is the minimum time after the beginning of the time range
  when an artifact excursion may be detected. Excursions before this time
  (i.e. very short excursions) are assumed to be spurious. In practice
  these either have strange shapes (hard to fit) or are excursions
  introduced by previous curve fits.

"min_first_detect_ms" is the minimum time after the beginning of the time
  range when the first artifact excursion may be detected. If the first
  attempt at detecting excursions finds an artifact before this time, the
  attempt is abandoned. This happens when the threshold is set too high and
  the tail of the exponential isn't detected.

(This is the end of the file.)

1.5 EXPGUESSPLOT.txt

Exponential fit debug plots are configured using a structure with the
following fields:

"fileprefix" is a prefix to use when building plot and report filenames.

"titleprefix" is a plot-safe human-readable prefix to use when building
  plot titles.

"plot_sizes_ms" is a cell array containing [min max] time ranges
  in milliseconds for which plots are to be generated.

"plot_labels" is a cell array containing filename- and plot-safe
  character vectors associated with each of the plotting time ranges.

"time_squash_ms" [min max] is a time range to set NaN when plotting
  (so that artifacts don't perturb the Y axis range), or [] to not squash.

Additional fields for nlFT_guessMultipleExpDecays:

"max_trial_count" is the maximum number of trials for which debug plots
  are to be generated.

"max_chan_count" is the maximum number of channels for which debug plots
  are to be generated.

(This is the end of the file.)

1.6 FOLDERMETA.txt

Metadata for data repository folders, ephys devices, signal banks within
ephys devices, and channels within signal banks are stored in a hierarchicial
set of structures. User-defined metadata may also be added to these
structures.

A "project metadata structure" is a structure containing top-level metadata
and metadata for each of the data folders used by the project. Folders
typically correspond to data captured by one specific piece of equipment or
using one specific software tool.

"folders" is a structure with one field per folder, indexed by user-assigned
  folder label, containing folder metadata structures.

User-defined metadata fields may also be added.

A "folder metadata structure" is a structure containing folder-level metadata
and metadata for each of the signal banks provided by the ephys device or
ephys software suite that created the folder. Signal banks may correspond to
hardware banks (such as specific headstages or groups of I/O channels) or to
virtual banks defined by the ephys software; the metadata representation is
the same.

The folder metadata structure has the following fields:

"path" is the filesystem path of this data folder.
"devicetype" is a label assigned by the LoopUtil library identifying the
  type of device that produced the data (i.e. identifying the helper functions
  needed to read that data).
"banks" is a structure with one field per signal bank, indexed by signal bank
  label, containing signal bank metadata structures.
"nativeorder" (optional) is a structure array containing bank and channel
  numbers in the order in which the underlying device sorted them. These
  are stored in "bank" and "channel" fields, respectively. This is not
  guaranteed to contain all types of channel.
"nativemeta" (optional) is an object containing device-specific metadata in
  its original form. This is typically, but not necessarily, a structure.

Other metadata fields specific to a given device type may also be present.
User-defined metadata fields may also be added.

In the event that one physical folder contains data from multiple ephys
devices, multiple folder metadata structures are produced. These will have
the same "path" value but different folder labels.

A "signal bank metadata structure" is a structure describing multiple
channels that are grouped within one real or virtual "bank" by an ephys
device or ephys software suite (for example, one headstage's analog recording
channels). The following fields are present:

"channels" is a vector containing (integer-valued) channel indices for which

```
  data is present. These are not guaranteed to be sorted or to be contiguous
  (sparse and unsorted channels are okay).
"samprate" is the data sampling rate in samples per second.
"sampcount" is the number of data samples in each channel's recording. This
  is assumed to be the same for all channels within a bank. For sparse series
  (event data), event timestamps are assumed to be in the range 1..sampcount.
"banktype" is a label assigned by the LoopUtil library identifying the type
  of signal contained within this bank. Defined types are:
  - 'analog' represents continuous-range time-varying signal data.
  - 'integer' represents discrete-range time-varying signal data.
  - 'boolean' represents a signal that is either "on" or "off". Changes may
    be interpreted as events.
  - 'flagvector' is integer data that represents multiple 'boolean' signals.
    Changes may be interpreted as events, and flags may be decoded using the
    "flagdefs" structure if present.
  - 'eventwords' is integer data stored as a sparse time series (event data
    rather than continuous data).
  - 'eventbool' is boolean data stored as a sparse time series (event data
    rather than continuous data).
"flagdefs" (optional) is present for banks of type 'flagvector'. It is a
  structure indexed by flag label containing the integer bit-mask values
  that correspond to each flag in the vector.
"nativetimetype" is the Matlab type name of the underlying "native" timestamp
  format. This is typically a sample count stored as a large integer type.
"nativedatatype" is the Matlab type name of the underlying "native" signal
  data.
"nativezerolevel" is the "native" data value corresponding to a signal value
  of zero. This is typically used with unsigned integer native data types.
"nativescale" is a multiplier used to convert "native" data values to
  floating-point data values in suitable units (volts, amperes, etc).
"fpunits" is a human-readable label indicating measurement units after
  conversion to floating point ('uV', 'V', 'uA', 'A', etc). This may be ''
  for data without meaningful units (integer, boolean, flag vector, etc).
"handle" is an opaque object used by the LoopUtil library to manage file
  I/O state. It typically contains file format details, file descriptors for
  open files, and state flags.
"nativemeta" (optional) is an object containing bank-specific metadata in
  the original device-supplied format. This is typically, but not
  necessarily, a structure.

Other metadata fields specific to a given bank type may also be present,
although this is typically stored in the "handle" object instead.
User-defined metadata fields may also be added.



This is the end of the file.
```

1.7 MODELPARAMSROBINSON.txt

```
The nlSynth_robinsonXXX functions accept a model parameters structure
with the fields described below.


Relevant references:

(Robinson 2002)
https://journals.aps.org/pre/abstract/10.1103/PhysRevE.65.041924

(Freyer 2011)
https://www.jneurosci.org/content/31/17/6353.short

(Hindriks 2023)
https://www.nature.com/articles/s42003-023-04648-x


Sigmoid parameters (common to all):

"qmax" is the maximum firing rate (1/sec); typically 250.
"threshlevel" is the average neuronal threshold (mV); typically 15.
"threshsigma" is the standard deviation of the neuronal threshold (mV);
  typically 6.0.


Neural dynamics parameters (common to all):

"alpha" is the inverse decay time (1/sec); typically 50.
"beta" is the inverse rise time (1/sec); typically 200.
"gamma" is the inverse within-cortex propagation time (1/sec); typically 100.


Cortico-thalamic circuit parameters (for SimulateHindriksNetwork):

"halfdelay_ms" is half of the round-trip cortex/thalamus loop time (ms);
  typically 40.


Additional coupling parameters (for SimulateHindriksNetwork):

"noisecoupling" is the coupling weight of noise into the specific nucleus.
  Typically 0.5.
"mixturecoupling" is the coupling weight of mixed-population cortex
  excitatory signals into the cortex. Typically 0.07.


Noise generation parameters (for SimulateHindriksNetwork):
```

```
"noisemean" is the mean noise signal value; typically 0.
"noisesigma" is the standard deviation of additive noise; typically 0.1.
"noisemultfactor" is "chi" in Freyer 2011/Hindriks 2023; the standard
  deviation of multiplicative noise is chi * noisesigma. Typically 0.3.



This is the end of the file.
```

1.8 PROCFUNC.txt

```
A processing function handle is called to perform signal processing when
iterating across input channels. The intention is to allow user-defined
signal processing while never having to load more than one input channel
into RAM at any given time.


A processing function has the form:

resultval = procfunc( metadata, folderid, bankid, chanid, ...
  wavedata, timedata, wavenative, timenative )

"metadata" is a project metadata structure, per FOLDERMETA.txt.
"folderid" is a character array with the user-defined folder label of the
  signal being processed.
"bankid" is a character array with the device-defined bank label of the
  signal being processed.
"chanid" is a scalar containing the channel number of the signal being
  processed.
"wavedata" is a vector containing signal samples being processed. Data values
  are floating-point values (typically volts).
"timedata" is a vector containing timestamps of the signal samples being
  processed. These are floating-point values (typically seconds).
"wavenative" is a vector containing signal samples being processed in "native"
  format (typically 16-bit signed or unsigned integers). Data scale is
  equipment-dependent (per FOLDERMETA.txt).
"timenative" is a vector containing timestamps of the signal samples in
  "native" format (typically 32- or 64-bit signed integer sample counts).

"resultval" is a user-defined object representing the results of processing
  one signal. For top-level processing, this is typically a structure
  containing feature information extracted from the signal waveform. For
  preprocessing, this is typically a filtered copy of the waveform itself.


A typical processing function definition would be as follows. This example
wraps a helper function that is passed additional arguments set at the time
the processing function is defined.
```

```
tuning_parameters = (stuff);
other_parameters = (stuff);
procfunc = @( metadata, folderid, bankid, chanid, ...
  wavedata, timedata, wavenative, timenative ) ...
  helper_do_processing( metadata, folderid, bankid, chanid, ...
    wavedata, timedata, tuning_parameters, other_parameters );


This is the end of the file.
```

1.9 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 2

"nlArt" Functions

2.1 nlArt_extendNaNQuartile.m

```
% function newseries = nlArt_extendNaNQuartile( oldseries, threshold )
%
% This segments a signal into NaN and non-NaN regions, and extends the NaN
% regions to cover adjacent samples that are sufficiently large excursions
% from their host non-NaN regions.
%
% This is intended to extend artifact-squash regions to cover portions of
% the artifact that are still present after squashing.
%
% "oldseries" is the series to process.
% "threshold" is the amount by which the signal must depart from the median
%   level to be considered a residual artifact. This is a multiple of the
%   median-to-quartile distance (about two thirds of a standard deviation).
%   Typical values are 6-12 for clear exponential excursions. If this is
%   NaN or Inf, no squashing is performed.
%
% "newseries" is a copy of "oldseries" with NaN regions extended to cover
%   adjacent excursions.
```

2.2 nlArt_fitExpDecay.m

```
% function fitparams = ...
%   nlArt_fitExpDecay( timeseries, waveseries, offset, method )
%
% This fits an exponential function with the form:
%   f(t) = coeff * exp( t/tau ) + offset
%
% This is intended for artifact curve-fitting.
%
```

```
% "timeseries" is a vector with sample timestamps.
% "waveseries" is a vector with sample values to be curve-fit.
% "offset" is a scalar specifying an offset value, [ tmin tmax ] specifying
%   a time range to estimate the offset value from, or [] or NaN to let this
%   function guess at the offset by whatever means it sees fit.
% "method" is a character vector specifying the algorithm to use to perform
%   the exponential fit.
%   'log' does a line fit in the logarithmic domain.
%   'pinmax' pins the curve fit at the most extreme end.
%   'pinboth' pins the curve fit at both ends and jointly optimizes tau and
%     offset.
%
% "fitparams" is a structure with curve fit parameters per ARTFITPARAMS.txt.
```

2.3 nlArt_getEndpointRamps.m

```
% function rampseries = nlArt_getEndpointRamps( dataseries, spanfrac )
%
% This segments a signal into NaN and non-NaN regions, and constructs a
% piecewise-linear background connecting the endpoints of all non-NaN
% regions (defined across the entire data series, including NaN regions).
%
% This is intended to be subtracted before filtering to suppress ringing
% from discontinuities in the signal (stepwise changes over a short time
% interval, or sudden changes in slope when interpolating over longer
% gaps). Since FFT-based filtering assumes the signal is periodic, this also
% suppresses edge effects (by forcing the signal endpoints to match).
%
% "dataseries" is the series to process.
% "spanfrac" is the fraction of the length of non-NaN spans to keep at each
%   endpoint when estimating the line fit (e.g. 0.05 to pay attention to the
%   first and last 5% of each non-NaN segment). This is intended to suppress
%   transient samples at the endpoints. If this is 0 or NaN, only the first
%   and last sample of each non-NaN region are used.
%
% "rampseries" is a piecewise-linear curve defined across the entire range
%   of samples, bridging the endpoints of all non-NaN segments.
```

2.4 nlArt_guessMultipleExpDecays.m

```
% function [ newwave fitlist reportstr ] = nlArt_guessMultipleExpDecays( ...
%   timeseries, waveseries, fitconfig, plotconfig, ...
%   tattle_verbosity, report_verbosity, reportlabel )
%
% This uses black magic to guess where exponential curve fits should be
% performed to remove artifacts. It's more robust than fitting to fixed
```

```
% time spans, but requires a lot of hand-tuned configuration parameters.
%
% Artifacts are assumed to be confined to a user-specified span. Fits are
% subtracted only within that span, for generating the corrected wave.
%
% Plots of curve fit attempts are optionally generated. These are useful
% when hand-tuning the configuration parameters.
%
% "timeseries" is a vector containing sample times.
% "waveseries" is a vector containing sample values to be curve-fit.
% "fitconfig" is a configuration structure, per EXPGUESSCONFIG.txt.
% "plotconfig" is a structure with the fields described in EXPGUESSPLOT.txt,
%   or struct([]) to suppress plotting and reports.
% "tattle_verbosity" is 'quiet', 'terse', 'normal', or 'verbose'. This
%   controls how many debugging/progress messages are sent to the console.
% "report_verbosity" is 'quiet', 'terse', 'normal', or 'verbose'. This
%   controls how many debugging/progress messages are appended to the
%   report string.
% "reportlabel" is an identifier to prepend to curve fit report messages.
%
% "newwave" is a copy of "waveseries" with the curve fits subtracted.
% "fitlist" is a cell array holding curve fit parameters for successive
%   curve fits, per ARTFITPARAMS.txt.
% "reportstr" is a character vector containing debugging/progress messages
%   selected by "report_verbosity".
```

2.5 nlArt_rampOverStimStep.m

```
% function newwave = ...
%   nlArt_rampOverStimStep( timeseries, waveseries, ramp_span, stim_span )
%
% This NaNs out a stimulation artifact region and applies a ramp over a
% larger portion of the wave to turn the stepwise level shift across the
% stimulation region into a more graceful change (to avoid filter ringing
% in subsequent processing steps).
%
% This is intended to be used after artifact cancellation, so that there
% aren't large excursions in the post-stimulation signal.
%
% "timeseries" is a vector with sample timestamps.
% "waveseries" is a vector with sample values.
% "ramp_span" [ min max ] is a time range over which to apply the ramp.
% "stim_span" [ min max ] is a time range containing stimulation artifacts
%   to be squashed, or [] to auto-detect an existing NaN span.
%
% "newwave" is a copy of "waveseries" with the stimulation region NaNed out
%   and a gradual ramp between pre-stimulation and post-stimulation DC
%   levels.
```

2.6 nlArt_reconFit.m

```
% function reconwave = nlArt_reconFit( timeseries, fitparams )
%
% This reconstructs one or more curve fits over a specified time span.
% This is intended for use with artifact curve-fitting functions.
%
% "timeseries" is a vector with sample timestamps.
% "fitparams" is a structure (with nlArt_XX curve fit parameters) per
%   ARTFITPARAMS.txt, or a "cfit" object with curve fit toolbox parameters,
%   or a cell array containing multiple such structures/objects.
%
% "reconwave" is a vector with reconstructed sample values.
```

2.7 nlArt_removeArtifactsSigma.m

```
% function newseries = nlArt_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.
%
% "newseries" is a copy of "oldseries" with artifacts replaced with NaN.
%
% 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.
```

2.8 nlArt_removeMultipleExpDecays.m

```
% function [ newwave fitlist ] = nlArt_removeMultipleExpDecays( ...
%  timeseries, waveseries, fenceposts, method )
%
% This attempts to curve fit and remove a settling artifact composed of
% multiple exponential decay tails.
%
% DC levels are discarded from the curve fits - the "offset" parameters in
% the curve fits are all set to zero.
%
% "timeseries" is a vector with sample timestamps.
% "waveseries" is a vector with sample values to be modified.
% "fenceposts" is a vector containing times to be used as span endpoints for
%   curve fitting. The span between the last two times is curve fit and its
%   contribution subtracted, and then the next-last span, and so forth. Only
%   the portion of the wave following the earliest fencepost is modified.
% "method", if present, is a character vector or cell array specifying the
%   algorithms to use to perform the exponential fits ('log', 'pinmax', or
%   'pinboth', per "nlArt_fitExpDecay()"). If this is a character vector,
%   the same algorithm is used for all fits. If this is a cell array, it
%   should have one fewer elements than "fenceposts", and specifies the
%   algorithm used for each fit. If "method" is absent, or if any method is
%   specified as '', a default method is chosen.
%
% "newwave" is a copy of "waveseries" with the curve fits subtracted.
% "fitlist" is a cell array holding curve fit parameters for successive
%   curve fits (farthest/slowest first). Curve fit parameters are structures
%   as described in ARTFITPARAMS.txt.
```

2.9 nlArt_stripDCFromFit.m

```
% function newfit = nlArt_stripDCFromFit( oldfit )
%
% This squashes the DC component of one or more curve fit descriptors.
%
% This only knows how to modify nlArt_XX curve fits. Curve fit toolbox
% curve fits are returned unchanged.
%
% "oldfit" is a structure with nlArt_XX curve fit parameters per
%   ARTFITPARAMS.txt, or a "cfit" object with curve fit toolbox parameters,
%   or a cell array containing multiple such structures/objects.
%
% "newfit" is a copy of "oldfit" modified to have DC components set to zero.
```

Chapter 3

"nlBasis" Functions

3.1 nlBasis_calcExplainedVariance.m

```
% function expvar = nlBasis_calcExplainedVariance( datavalues, basis )
%
% This function estimates the explained variance of a basis vector model
% reconstruction of supplied data.
%
% For well-behaved distributions, the explained variance fraction is the
% square of the correlation coefficient of the original and reconstructed
% signals.
%
% This function calculates the mean across Nvectors in the original data
% and subtracts it from the original and reconstruction when calculating
% the variance. The mean would otherwise greatly inflate the explained
% variance.
%
% "datavalues" is a Nvectors x Ntimesamples matrix of sample vectors. For
%   ephys data, this is typically Nchans x Ntimesamples.
% "basis" is a structure describing a basis vector decomposition of the
%   data, per BASISVECTORS.txt.
%
% "expvar" is the fraction of the variance explained by the decomposition.
```

3.2 nlBasis_decomposeSignalsUsingBasis.m

```
% function [ coeffs residues ] = ...
%   nlBasis_decomposeSignalsUsingBasis( datavectors, basisvecs )
%
% This attempts to decompose one or more sample vectors using a previously
% extracted set of basis vectors (per BASISVECTORS.txt).
%
```

```
% NOTE - This will only give optimal output if the basis vectors are
% orthogonal! For non-orthogonal basis vectors, output will be valid but
% not necessarily minimum-energy.
%
% "datavectors" is a Nvectors x Ntimesamples matrix of sample vectors. For
%   ephys data, this is typically Nchans x Ntimesamples.
% "basisvecs" is a Nbasis x Ntimesamples matrix where each row is a basis
%   vector.
%
% "coeffs" is a Nvectors x Nbasis matrix with basis vector weights for each
%   input vector.
% "residues" is a Nvectors x Ntimesamples matrix containing the components of
%   the input vectors that could not be represented as a linear combination
%   of the basis vectors.
%
% (coeffs * basisvecs) + (residues) is an estimate of (datavectors).
```

3.3 nlBasis_estimateBasisFromCoefficients.m

```
% function newbasis = nlBasis_estimateBasisFromCoefficients( ...
%   datavalues, coeffs, bgmethod )
%
% This estimates basis vectors and background for a given set of signals,
% given a coefficient matrix.
%
% This is intended to be used to ensure consistent basis decompositions
% in situations where several different data matrices are derived from the
% same dataset. One data matrix is chosen as "canon" and decomposed, and
% the basis vectors in the other matrices are estimated using this function
% and the "canon" coefficients.
%
% "datavalues" is a Nvectors x Ntimesamples matrix of sample vectors. For
%   ephys data, this is typicaly Nchans x Ntimesamples.
% "coeffs" is a Nvectors x Nbasis matrix with basis vector weights for each
%   input vector, per BASISVECTORS.txt.
% "bgmethod" is 'zero' (background set to zero), 'average' (background set
%   to the average across channels), or 'basis' (background estimated by
%   treating it as an additional basis with weight 1 in all data vectors).
%
% "newbasis" is a basis description structure per BASISVECTORS.txt containing
%   a copy of "coeffs" along with estimated basis band background vectors.
```

3.4 nlBasis_getBackgroundCommon.m

```
% function newbasis = nlBasis_getBackgroundCommon( oldbasis )
%
```

```
% This takes a basis decomposition structure and moves components that are
% common to all inputs to the background vector. The coefficient matrix is
% adjusted to remove these background components from the signal vectors.
%
% This works by calling nlBasis_getBackgroundResidue() to move all basis
% components to the signal vectors, and then taking the minimum or maximum
% coefficient value for each basis as that basis's contribution to the
% common background.
%
% "oldbasis" is a structure defining the basis decomposition, per
%   BASISVECTORS.txt.
%
% "newbasis" is a modified version of "oldbasis" with the same basis vectors
%   but with weight coefficients and the background modified.
```

3.5 nlBasis_getBackgroundResidue.m

```
% function newbasis = nlBasis_getBackgroundResidue( oldbasis )
%
% This takes a basis decomposition structure and removes all basis components
% from the background vector. The coefficient matrix is adjusted to add these
% background components to the signal vectors.
%
% "oldbasis" is a structure defining the basis decomposition, per
%   BASISVECTORS.txt.
%
% "newbasis" is a modified version of "oldbasis" with the same basis vectors
%   but with weight coefficients and the background modified.
```

3.6 nlBasis_getBasisByName.m

```
% function [ fomvalue basis ] = nlBasis_getBasisByName( ...
%   datavalues, basiscounts, minfom, method, verbosity )
%
% This calls nlBasis_getBasisXX() to get a basis vector decomposition of
% input data by a user-specified method.
%
% "datavalues" is a Nvectors x Ntimesamples matrix of sample vectors. For
%   ephys data, this is typically Nchans x Ntimesamples.
% "basiscounts" is a vector containing basis component counts to test.
% "minfom" is the minimum figure of merit value to accept. If this is NaN,
%   the component count with the maximum figure of merit value is chosen. If
%   this is not NaN, then the smallest component count with a figure of
%   merit above minfom is chosen (or the largest figure of merit value if
%   none are above-threshold).
% "method" is 'kmeans', 'pca', 'ica_direct', or 'ica_pca'.
```

```
% "verbosity" is 'verbose', 'normal', or 'quiet'.
%
% "fomvalue" is a figure-of-merit value (typically fraction of explained
%   variance).
% "basis" is a structure describing the decomposition, per BASISVECTORS.txt.
```

3.7 nlBasis_getBasisDirectICA.m

```
% function [ expvar basis ] = nlBasis_getBasisDirectICA( ...
%   datavalues, basiscounts, minexpvar, verbosity )
%
% This performs independent component analysis of a set of input vectors and
% expresses the result as a basis vector decomposition per BASISVECTORS.txt.
%
% This performs ICA directly on the input (without performing PCA first).
%
% Mean subtraction is not performed; the mean across channels is treated as
% part of the signal rather than as background.
%
% NOTE - Performing ICA on raw time-series waveforms takes a while.
%
% "datavalues" is a Nvectors x Ntimesamples matrix of sample vectors. For
%   ephys data, this is typically Nchans x Ntimesamples.
% "basiscounts" is a vector containing independent component counts to test.
% "minexpvar" is the minimum explained variance to accept. If this is NaN,
%   the component count with the maximum explained variance is chosen. If
%   this is not NaN, then the smallest component count with an explained
%   variance above minexpvar is chosen (or the largest explained variance
%   if none are above-threshold).
% "verbosity" is 'verbose', 'normal', or 'quiet'.
%
% "expvar" is the explained variance of the ICA decomposition.
% "basis" is a structure describing the ICA decomposition, per
%   BASISVECTORS.txt. The background is zero.
```

3.8 nlBasis_getBasisKmeans.m

```
% function [ silval basis ] = nlBasis_getBasisKmeans( ...
%   datavalues, kvalues, minsilval, verbosity )
%
% This performs k-means decomposition of a set of input vectors and expresses
% the result as a basis vector decomposition with each mean being one basis.
%
% Basis vector decompositions are described in BASISVECTORS.txt. Expressing
% k-means in this form is intended to make it easier to use various auxiliary
% functions with the resulting decomposition. Coefficients are 1 for the each
```

```
% input vector's cluster mean and 0 otherwise, with a background of zero
% (so this doesn't actually do a real basis decomposition of the input).
%
% Silhouette values range from -1 to +1. A "good" decomposition has a
% silhouette value of at least 0.6.
%
% "datavalues" is a Nvectors x Ntimesamples matrix of sample vectors. For
%   ephys data, this is typically Nchans x Ntimesamples.
% "kvalues" is a vector containing cluster counts to test.
% "minsilval" is the minimum silhouette value to accept. If this is NaN,
%   all k values are tested and the one with the maximum silhouette value is
%   chosen. If this is not NaN, then the smallest k value with a silhouette
%   value of at least minsilval is chosen (or the largest silhouette value
%   if none are above-threshold).
% "verbosity" is 'normal' or 'quiet'.
%
% "silval" is the silhouette value of the k-means decomposition.
% "basis" is a structure describing the k-means decomposition, per
%   BASISVECTORS.txt. Each mean is one basis vector. The background is zero.
```

3.9 nlBasis_getBasisPCA.m

```
% function [ expvar basis ] = nlBasis_getBasisPCA( ...
%   datavalues, basiscounts, minexpvar, verbosity )
%
% This performs principal component analysis of a set of input vectors and
% expresses the result as a basis vector decomposition per BASISVECTORS.txt.
%
% The mean (which is removed by PCA) is saved as the "background" in the
% decomposition.
%
% "datavalues" is a Nvectors x Ntimesamples matrix of sample vectors. For
%   ephys data, this is typically Nchans x Ntimesamples.
% "basiscounts" is a vector containing principal component counts to test.
% "minexpvar" is the minimum explained variance to accept. If this is NaN,
%   the component count with the maximum explained variance is chosen. If
%   this is not NaN, then the smallest component count with an explained
%   variance above minexpvar is chosen (or the largest explained variance
%   if none are above-threshold).
% "verbosity" is 'normal' or 'quiet'.
%
% "expvar" is the explained variance of the PCA decomposition.
% "basis" is a structure describing the PCA decomposition, per
%   BASISVECTORS.txt. The mean is saved as the background.
```

3.10 nlBasis_getBasisPCAICA.m

```
% function [ expvar basis ] = nlBasis_getBasisPCAICA( ...
%   datavalues, basiscounts, minexpvar, verbosity, pcamaxbasis, pcaminexpvar )
%
% This performs principal component analysis of a set of input vectors to get
% an intermediate representation of the input, and then performs independent
% component analysis in PCA space. The resulting basis vectors are transformed
% back into signal space and expressed as a basis vector decomposition per
% BASISVECTORS.txt.
%
% The mean (which is removed by PCA) is saved as the "background" in the
% decomposition.
%
% "datavalues" is a Nvectors x Ntimesamples matrix of sample vectors. For
%   ephys data, this is typically Nchans x Ntimesamples.
%
% "basiscounts" is a vector containing independent component counts to test.
% "minexpvar" is the minimum explained variance to accept. If this is NaN,
%   the component count with the maximum explained variance is chosen. If
%   this is not NaN, then the smallest component count with an explained
%   variance above minexpvar is chosen (or the largest explained variance
%   if none are above-threshold).
% "verbosity" is 'verbose', 'normal', or 'quiet'.
% "pcamaxbasis" is the maximum number of PCA components to use in the first
%   step. If NaN or omitted, this defaults to 20.
% "pcaminexpvar" is the minimum explained variance to accept in the PCA
%   decomposition. If NaN or omitted, this defaults to 0.98. If this target
%   can't be met, the maximum number of PCA components is used.
%
% "expvar" is the explained variance of the PCA-ICA decomposition.
% "basis" is a structure describing the PCA-ICA decomposition, per
%   BASISVECTORS.txt. The mean is saved as the background.
```

Chapter 4

"nlIO" Functions

4.1 nlIO_filterChanList.m

```
% function newlist = nlIO_filterChanList( oldlist, filterrules )
%
% This function filters a channel list, keeping or discarding entries
% according to user-specified filter rules.
%
% "oldlist" is a channel list to filter, per "CHANLIST.txt".
% "filterrules" is a structure with zero or more of the following fields:
%   "keepfolders" is a cell array containing regex patterns that folder labels
%     must match.
%   "omitfolders" is a cell array containing regex patterns that folder labels
%     must not match.
%   "keepbanks" is a cell array containing regex patterns that bank labels
%     must match.
%   "omitbanks" is a cell array containing regex patterns that bank labels
%     must not match.
%   "keepchans" is a vector containing channel indices that should be kept.
%     Any channels not in this list are discarded.
%   "omitchans" is a vector containing channel indices that must be discarded.
%   "keepbanktypes" is a cell array containing bank type identifiers that
%     should be matched. Any bank type identifiers not in this list are
%     discarded.
%   "omitbanktypes" is a cell array containing bank type identifiers that
%     must not be matched.
%
% "newlist" is a subset of "oldlist" containing entries that pass all rules.
```

4.2 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 IDs or labels.
% "format" is a "sprintf" conversion format for turning a candidate ID or
%   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.
```

4.3 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.
```

4.4 nlIO_getChanListFromMetadata.m

```
% function chanlist = nlIO_getChanListFromMetadata(metadata)
%
% This generates a channel list describing all channels defined in the
% specified metadata structure. The "banktype" field is copied to the
% channel list scalar metadata structure to facilitate list filtering.
%
% "metadata" is a project metadata structure, per "FOLDERMETA.txt".
%
% "chanlist" is a channel list, per "CHANLIST.txt".
```

4.5 nlIO_iterateChannels.m

```
% function resultvals = ....
%   nlIO_iterateChannels(metadata, chanlist, memchans, procfunc)
%
% This iterates through a set of channels, loading each channel's waveform
% data in sequence and calling a processing function with that data.
% Processing output is aggregated and returned.
%
% This is implemented such that only a few channels are loaded at a time.
%
% Channel time series are stored as sample numbers (not times). Analog data
% is converted to microvolts. TTL data is converted to boolean.
%
% "metadata" is a project metadata structure, per FOLDERMETA.txt.
% "chanlist" is a structure listing channels to process, per CHANLIST.txt.
% "memchans" is the maximum number of channels that may be loaded into
%   memory at the same time.
% "procfunc" is a function handle used to transform channel waveform data
%   into "result" data, per PROCFUNC.txt.
%
% "resultvals" is a channel list structure that has bank-level channel lists
%   augmented with a "resultlist" field, per CHANLIST.txt. The "resultlist"
%   field is a cell array containing per-channel output from "procfunc".
```

4.6 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.
```

4.7 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.
```

4.8 nlIO_readBinaryFile.m

```
% function [is_ok sampdata] = nlIO_readBinaryFile(fname, dtype, samprange)
%
% This attempts to read a packed array of the specified data type from the
% specified file.
%
% NOTE - The data is returned as-is, _not_ promoted to double.
%
% "fname" is the name of the file to read from.
% "dtype" is a string identifying the Matlab data type (e.g. 'uint32').
```

```
% "samprange" [first last] is the range of data samples (not bytes) to read,
%   specified with Matlab's conventions (the first sample is sample 1, not 0).
%   Specify an empty sample range ([]) to read all data in the file.
%
% "is_ok" is set to true if the operation succeeds and false otherwise.
% "sampdata" is an array containing the sample values, in native format.
```

4.9 nlIO_readFolderMetadata.m

```
% function [ isok newmeta ] = ...
%   nlIO_readFolderMetadata( oldmeta, newlabel, indir, devicetype )
%
% This probes the specified directory, looking for data and metadata files
% from the specified type of ephys machine or software suite. If the type
% is given as 'auto', this searches for types that it knows how to identify.
%
% Metadata structure format is defined in "FOLDERMETA.txt".
%
% "oldmeta" is a project metadata structure to add to. Pass an empty structure
%   to create a new project metadata structure.
% "newlabel" is used as a folder label for adding this folder's metadata to
%   the project metadata structure.
% "indir" is the directory to search.
% "devicetype" is a character array specifying the type of architecture to
%   look for. Known types are 'intan' and 'openephys'. Use 'auto' for
%   automatic detection.
%
% "isok" is true if folder metadata was successfully read and false otherwise.
% "newmeta" is a copy of "oldmeta" with new folder metadata added. If no new
%   metadata was found, a copy of "oldmeta" is still returned. If metadata
%   from multiple devices is found, multiple folder metadata structures are
%   added. "makeUniqueStrings" is called to avoid folder label conflicts.
```

4.10 nlIO_searchForDir.m

```
% function dirs_found = nlIO_searchForDir( startdir, targetname )
%
% This searches a directory tree for folders that match the specified name.
% This wraps "dir", so the folder name can contain wildcards.
%
% "startdir" is the top-level folder to look in.
% "targetname" is the folder name to match. This is passed to "dir", so it
%   can contain wildcards.
%
% "dirs_found" is a cell array containing paths to folders that match the
%   target name.
```

4.11 nlIO_searchForFile.m

```
% function [ fnames_found paths_found ] = ...
%   nlIO_searchForFile( startdir, targetname )
%
% This searches a directory tree for files that match the specified name.
% This wraps "dir", so filenames can contain wildcards.
%
% "startdir" is the top-level folder to look in.
% "targetname" is the file name to match. This is passed to "dir", so it can
%   contain wildcards.
%
% "fnames_found" is a cell array containing the names of files found, without
%   paths.
% "paths_found" is a cell array containing the paths of files found, without
%   filenames.
```

4.12 nlIO_subtractFromChanList.m

```
% function newlist = nlIO_subtractFromChanList(oldlist, removelist)
%
% This function removes the specified members from a channel list, if the
% members are present.
%
% "oldlist" is the channel list (per "CHANLIST.txt").
% "removelist" is a channel list containing members to be removed.
%
% "newlist" is a copy of "oldlist" that does not contain any members that
%   were listed in "removelist".
```

4.13 nlIO_writeBinaryFile.m

```
% function is_ok = 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.
```

4.14 nlIO_writeTextFile.m

```
% function is_ok = nlIO_writeTextFile(fname, textcontent)
%
% This attempts to write the specified character vector as ASCII text.
% This is a wrapper for "nlIO_writeBinaryData" with type 'char*1'.
%
% "fname" is the name of the file to write to.
% "textcontent" is a character vector containing the text to write.
%
% "is_ok" is true if the operation succeeds and false otherwise.
```

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_axesPlotStripChart.m

```
% function nlPlot_axesPlotStripChart( thisax, timeseries, wavematrix, ...
%   wavelabels, colstart, colspread, wavestyle, want_axis_labels, ...
%   timerange, wave_yrange, wave_yspacing, hcursorlist, hcursorcol, ...
%   vcursorlist, vcursorcol, xtitle, ytitle, legendpos, figtitle )
%
% This plots a series of waveforms stacked vertically, strip-chart style.
%
% This is intended to be called multiple times, so that several sets of
% waves and cursors can be added to the same plot axes.
%
% "thisax" is the "axes" object to render to.
% "timeseries" is a vector containing sample times.
% "wavematrix" is a Nchans x Nsamples matrix containing data to plot.
% "wavelabels" is a cell array of length Nchans containing legend labels
%   for each wave. These are also plotted on the Y axis if requested. Empty
%   labels ('') suppress a given wave's label.
% "colstart" is a [ r g b ] colour triplet specifying the colour of the
%   first wave plotted.
```

```
% "colspread" is the total angle to walk around the colour wheel while
%   plotting waves, in degrees.
% "wavestyle" is a character vector with the line style to use when plotting.
% "want_axis_labels" is true to render each wave's label next to the Y axis
%   in addition to in the legend, false otherwise.
% "timerange" [ min max ] is the time span to render, or [] to auto-range.
% "wave_yrange" [ min max ] is the Y range to make available for each wave,
%   or [] to auto-range.
% "wave_yspacing" is the Y distance to offset successive waves in the plot,
%   or [] or NaN to auto-range.
% "hcursorlist" is a vector containing Y coordinates for horizontal cursors.
% "hcursorcol" [ r g b ] is the colour to use when rendering horizontal
%   cursors. If there are no cursors, this may be [] or NaN.
% "vcursorlist" is a vector containing X coordinates for vertical cursors.
% "vcursorcol" [ r g b ] is the colour to use when rendering vertical
%   cursors. If there are no cursors, this may be [] or NaN.
% "xtitle" is the title to give to the X axis, or '' to not modify the title.
% "ytitle" is the title to give to the Y axis, or '' to not modify the title.
% "legendpos" is a position specifier to pass to the "legend" command, or
%   'off' to remove the legend, or '' to not alter the legend location.
% "figtitle" is the title to apply to the figure, or '' to not alter the
%   title.
%
% No return value.
```

5.5 nlPlot_axesPlotSurface2D.m

```
% function nlPlot_axesPlotSurface2D( thisax, zdata, xvalues, yvalues, ...
%   xrange, yrange, xloglin, yloglin, zloglin, xtitle, ytitle, figtitle )
%
% This plots a 2D array of data values as a heatmap or spectrogram style
% plot.
%
% "thisax" is the "axes" object to render to.
% "zdata" is a matrix indexed by (y,x) of data values to plot.
% "xvalues" is a series of X coordinate values corresponding to each column
%   of zdata. If there are as many values as columns, they're bin midpoints.
%   If there's one more value than there are columns, they're bin edges.
% "yvalues" is a series of Y coordinate values corresponding to each row of
%   zdata. If there are as many values as rows, they're bin midpoints. If
%   there's one more value than there are rows, they're bin edges.
% "xrange" [ min max ] is the range of X values to render, or [] for auto.
% "yrange" [ min max ] is the range of Y values to render, or [] for auto.
% "xloglin" is 'log' or 'linear', specifying the X axis scale.
% "yloglin" is 'log' or 'linear', specifying the Y axis scale.
% "zloglin" is 'log' or 'linear', specifying whether to log-compress zdata.
% "xtitle" is the title to use for the X axis, or '' to not set one.
% "ytitle" is the title to use for the Y axis, or '' to not set one.
```

```
% "figtitle" is the title to use for the figure, or '' to not set one.
%
% No return value.
```

5.6 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.
```

5.7 nlPlot_getColorMapHotCold.m

```
% function newmap = nlPlot_getColorMapHotCold( coldcolor, hotcolor, exponent )
%
% This function returns a Matlab colormap that spans from the "cold" colour
% (negative values), to black, to the "hot" colour (positive values).
%
% This is intended to be used with a "clim" range centred on zero (i.e.
% symmetric).
%
% "coldcolor" [ r g b ] is the colour to use for the most negative end of
%   the colormap. Components are in the range 0..1.
% "hotcolor" [ r g b ] is the colour to use for the most positive end of
%   the colormap. Components are in the range 0..1.
% "exponent" determines the slope of the colour gradient. A value of 1 gives
%   a linear gradient. Values closer to 0 make near-zero data values easier
%   to distinguish.
%
% "newmap" is a Matlab colormap.
```

5.8 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.
```

```
% Additional colours are "blk", "wht", and "gry".
%
% "cols" is a structure containing color triplets.
```

5.9 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.
```

5.10 nlPlot_getLineMarkStyleSpread.m

```
% function [ linestyles markstyles ] = nlPlot_getLineMarkStyleSpread( count )
%
% This function returns a list of Matlab plotting line style and marker style
% specifiers, chosen to avoid repeating combinations where possible.
%
% "count" is the number of entries to return.
%
% "linestyles" is a cell array containing character vectors to be passed
%   as 'LineStyle' plot arguments.
% "markstyles" is a cell array containing character vectors to be passed as
%   'Marker' plot arguments.
```

5.11 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.
```

5.12 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.
```

5.13 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

"nlProc" Functions

6.1 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.
```

6.2 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.
```

6.3 nlProc_calcCircularStats.m

```
% function [ cmean cvar lindev ] = nlProc_calcCircularStats( angleseries )
%
% This calculates the circular mean, circular variance, and linear standard
% deviation for a specified series of angles.
%
% "angleseries" is a vector containing angles in radians.
%
% "cmean" is the circular mean of the angle series.
% "cvar" is the circular variance of the angle series. Phase locking value
%   is (1 - cvar).
% "lindev" is the linear standard deviation of the angle series. For tightly
%   clustered angles, this can be more intuitive than circular variance.
```

6.4 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%) ].
```

6.5 nlProc_calcSmoothedRMS.m

```
% function [ smoothed_rms smoothed_rms_lagged ] = ...
%   nlProc_calcSmoothedRMS( wavedata, tausamples )
%
% This computes the square of the input signal, smooths it, and returns
% the square root of the smoothed squared signal.
%
% Smoothing is done by constructing a first-order exponential filter and
% applying it backwards and forwards in time using "filtfilt". This gets
% around some of Matlab's oddities with very-low-frequency filters.
%
% The "lagged" output uses "filter" to apply the exponential filter forwards
% in time only (producing causal output).
%
% "wavedata" is a vector containing the sample series to process.
% "tausamples" is the smoothing time constant in samples.
%
% "smoothed_rms" is the square root of the smoothed squared signal with
%   acausal (bidirectional) filtering.
% "smoothed_rms_lagged" is the square root of the smoothed squared signal
%   with causal (one-direction) filtering.
```

6.6 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.
```

6.7 nlProc_calcSteppedWindowSpectrogram.m

```
% function [ freqlist timelist spectpowers ] = ...
%   nlProc_calcSteppedWindowSpectrogram( ...
%     timeseries, waveseries, winsize, winstep, timespan, freqspan )
%
% This computes a spectrogram of a signal using a stepped rectangular
% window.
%
% The waveform is assumed to include the requested time span.
%
% "timeseries" is a vector containing sample timestamps.
% "waveseries" is a vector containing waveform values.
% "winsize" is the window duration in seconds.
% "winstep" is the window step distance in seconds.
% "timespan" [ min max ] is the time region within which the window is to
%   be stepped.
% "freqspan" [ min max ] is the range of frequencies to evaluate.
%
% "freqlist" is a vector containing frequencies for which power was computed.
```

```
% "timelist" is a vector containing window center times that were evaluated.
% "spectpowers" is a nFrequencies x nTimes matrix containing evaluated
%   spectral power (in arbitrary units).
```

6.8 nlProc_compareFlagSeries.m

```
% function [ message fp tp fn ] = ...
%   nlProc_compareFlagSeries( testflags, correctflags )
%
% This function compares two boolean signals' time series, computing
% confusion matrix statistics and generating a report.
%
% "testflags" is a vector of boolean values to evaluate.
% "correctflags" is a vector of boolean values representing ground truth for
%   the test series.
%
% "message" is a chacacter array containing a human-readable summary of the
%   confusion matrix statistics.
% "fp" is the number of false-positive samples (test without correct).
% "tp" is the number of true-positive samples (test and correct together).
% "fn" is the number of false-negative samples (correct without test).
```

6.9 nlProc_computeAverageSignal.m

```
% function avgseries = ...
%   nlProc_computeAverageSignal(metadata, chanlist, memchans, preprocfunc)
%
% This iterates through a list of channels, computing the average signal
% value of all specified channels. The average signal is returned.
%
% NOTE - It is the user's responsibility to ensure that all listed channels
% are time-aligned and have data values that use the same scale. If all
% signals are from the same ephys machine, that's usually handled. Otherwise
% the preprocessing function should handle that.
%
% "metadata" is the project metadata structure, per FOLDERMETA.txt.
% "chanlist" is a structure listing the channels to be averaged, per
%   CHANLIST.txt.
% "memchans" is the maximum number of channels that may be loaded into
%   memory at the same time.
% "preprocfunc" is a function handle that is called to preprocess each
%   channel's data prior to being averaged, per PROCFUNC.txt. This typically
%   performs artifact removal (filtering happens after re-referencing).
%
% "avgseries" is a data series computed as the average of the input channels.
%   Input series of different lengths are tolerated, but all are assumed to
```

```
%   start at the same time and to have the same sampling rate.
```

6.10 nlProc_deglitchCount.m

```
% function newseries = ...
%   nlProc_deglitchCount( oldseries, glitchsamps, dropoutsamps)
%
% This function removes spurious gaps (drop-outs) and brief events (glitches)
% from a one-dimensional boolean vector.
%
% This version of the function specifies durations using sample counts.
%
% "oldseries" is a logical vector to process.
% "glitchsamps" is the longest event duration to reject as spurious.
% "dropoutsamps" is the longest gap duration to reject as spurious.
%
% "newseries" is a modified version of "oldseries".
```

6.11 nlProc_deglitchTime.m

```
% function newseries = ...
%   nlProc_deglitchTime( oldseries, samprate, glitchtime, dropouttime)
%
% This function removes spurious gaps (drop-outs) and brief events (glitches)
% from a one-dimensional boolean vector.
%
% This version of the function specifies durations in seconds.
%
% "oldseries" is a logical vector to process.
% "samprate" is the sampling rate of the input signal.
% "glitchtime" is the longest event duration to reject as spurious.
% "dropouttime" is the longest gap duration to reject as spurious.
%
% "newseries" is a modified version of "oldseries".
```

6.12 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.
```

6.13 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.
```

6.14 nlProc_examineLFPSpectrum.m

```
% function [ isgood typelabel fitexponent ...
%   spectpowers spectfreqs fitpowers ] = nlProc_examineLFPSpectrum( ...
%     wavedata, samprate, freqrange, binwidth )
%
% This takes the power spectrum of the specified signal, bins it in the
% log-frequency domain, rejects narrow peaks, and then tries to fit a power
% law curve to it. If this looks like pink noise or red noise, it's a valid
% LFP; if this looks like white noise or like hash, it isn't.
%
% FIXME - In a perfect world we'd evaluate "burstiness" and give different
```

```
% labels based on whether a valid LFP was quiet or bursty. That's NYI.
%
% "wavedata" is a vector containing waveform sample data.
% "samprate" is the sampling rate of the waveform data.
% "freqrange" [ min max ] is the range of frequencies to fit over.
% "binwidth" is the relative width of frequency bins. A value of 0.1 would
%   mean a bin width of 2 Hz for a bin with a center frequency of 20 Hz.
%
% "isgood" is true if the spectrum looks like LFP background, false otherwise.
% "typelabel" is a human-readable descriptive label. Typical values are
%   'lfp', 'lfpbad', 'whitenoise', 'powerlaw', and 'hash'.
% "fitexponent" is the exponent of the power-law fit. Pink noise is -1, red
%   noise is -2.
% "spectpowers" are the binned power spectrum powers (linear, not dB).
% "spectfreqs" are the spectrum bin center frequencies in Hz.
% "fitpowers" are the curve-fit power law powers at the bin frequencies.
```

6.15 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.
```

6.16 nlProc_fillNaNRows.m

```
% function newmatrix = nlProc_fillNaNRows( oldmatrix )
%
% This accepts a Nvectors x Ntimesamples matrix and calls nlProc_fillNaN to
% fill gaps in each of the matrix rows.
%
% "oldmatrix" is a Nvectors x Ntimesamples matrix containing NaN segments.
%
% "newmatrix" is an interpolated copy of "oldmatrix" without NaN segments.
```

6.17 nlProc_filterBrickBandStop.m

```
% function newwave = nlProc_filterBrickBandStop( oldwave, samprate, bandlist )
%
% This performs band-stop filtering in the frequency domain by squashing
% frequency components (a "brick wall" filter). This causes ringing near
% large disturbances (a top-hat in the frequency domain gives a sinc
% function impulse response).
%
% "oldwave" is the signal to filter. This is assumed to be real.
% "samprate" is the sampling rate of "oldwave".
% "bandlist" is a cell array containing [ min max ] tuples indicating
%   frequency ranges to squash.
%
% "newwave" is a filtered version of "oldwave".
```

6.18 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.
```

6.19 nlProc_findCorrelatedChannels.m

```
% function [ changroups rvalues groupdefs ] = ...
%   nlProc_findCorrelatedChannels( wavedata, thresh_abs, thresh_rel )
%
% This attempts to find sets of strongly-correlated channels in waveform data.
% These may be floating channels coupling identical noise (for high-frequency
% data) or may be measuring from the same environment (for LFP data).
%
% NOTE - Channel correlation time goes up as the square of the number of
% channels!
%
% Correlation is judged using Pearson's Correlation Coefficient.
%
% "wavedata" is an Nchans*Nsamples matrix containing waveform data.
% "thresh_abs" is an absolute threshold. Channel pairs with correlation
%   coefficients above +thresh_abs are assumed to be copies.
%   NOTE - Differential channel pairs with have coefficients below -thresh_abs;
%   this is okay, and gets taken into account for thresh_rel per below.
% "thresh_rel" is a relative threshold. Channel pairs with correlation
%   coefficients above this multiple of a "typical" correlation coefficient
%   value are assumed to be copies. The "typical" value is the median of the
%   absolute value of all correlation coefficients that are below +thresh_abs
%   and above -thresh_abs.
%
% "changroups" is a vector indicating which group each channel is a member of,
%   or NaN if a channel is not a member of a group (not strongly correlated).
% "rvalues" is an Nchans*Nchans matrix containing correlation coefficient
%   values for all channel pairs.
% "groupdefs" is a cell array containing vectors representing groups of
%   mutually correlated channels. Each vector contains channel indices for
%   the members of that group.
```

6.20 nlProc_findNaNSpans.m

```
% function [ validstart validend nanstart nanend ] = ...
%   nlProc_findNaNSpans( dataseries )
%
% This segments a signal into NaN and non-NaN regions.
%
% "dataseries" is a vector containing signal samples.
%
% "validstart" is a vector containing the starting sample indices of
%   non-NaN regions.
% "validend" is a vector containing the ending sample indices of non-NaN
%   regions.
% "nanstart" is a vector containing the starting sample indices of NaN
%   regions.
% "nanend" is a vector containing the ending sample indices of NaN regions.
```

6.21 nlProc_findSpectrumPeaks.m

```
% function [ peakfreqs peakheights peakwidths binlevels bincenters ] = ...
%   nlProc_findSpectrumPeaks( ...
%     wavedata, samprate, peakwidth, backgroundwidth, peakthresh )
%
% This takes the frequency spectrum of the specified signal, bins it in the
% log-frequency domain, and looks for narrow peaks against the background.
%
% "wavedata" is a vector containing waveform sample data.
% "samprate" is the sampling rate of the waveform data.
% "peakwidth" is the relative width of the fine-resolution frequency bins.
%   A value of 0.1 would mean a bin width of 2 Hz at a frequency of 20 Hz.
% "backgroundwidth" is the ratio between the upper and lower frequencies of
%   the span used to evaluate noise background around any given bin. A value
%   of 2.0 would mean evaluating noise over a one-octave span.
% "peakthresh" is the magnitude threshold for recognizing a peak in the
%   frequency spectrum. This is a multiple of the average local background.
%
% "peakfreqs" is a vector containing peak center frequencies in Hz.
% "peakheights" is a vector containing peak heights relative to the background.
% "peakwidths" is a vector containing relative peak widths (FWHM / frequency).
% "binmags" is a vector containing frequency spectrum bin magnitudes.
% "binfreqs" is a vector containing frequency spectrum bin center frequencies.
```

6.22 nlProc_fitCosine.m

```
% function [ fitmag fitfreq fitphase fitpoly ] = nlProc_fitCosine( ...
```

```
%   wavedata, samprate, freqrange, polyorder )
%
% This curve-fits a constant-amplitude cosine to the specified input wave.
% This is intended to be used when the approximate frequency is known.
%
% A polynomial-fit background is optionally subtracted before the cosine fit
% is performed. If unspecified, order 0 is used (mean subtraction).
%
% "wavedata" is the waveform data series to curve fit.
% "samprate" is the sampling rate of the waveform.
% "freqrange" [ min max ] is the frequency range to curve fit across.
% "polyorder" (optional) is the polynomial fit order to use before doing the
%   cosine fit. This defaults to 0th order (mean subtraction).
%
% "fitmag" is the amplitude of the curve-fit cosine wave.
% "fitfreq" is the frequency of the curve-fit cosine wave.
% "fitphase" is the phase of the curve-fit cosine wave at the first sample
%   of the input wave.
% "fitpoly" is a row vector containing polynomial fit coefficients, highest
%   order first. For 0th order (default), this is a scalar containing the
%   mean of the input signal.
```

6.23 nlProc_getBinEdgesFromMidpoints.m

```
% function edgelist = nlProc_getBinEdgesFromMidpoints( midlist, scaletype )
%
% This makes reasonable guesses at histogram bin edges given a list of bin
% midpoints.
%
% "midlist" is a vector containing bin midpoint values.
% "scaletype" is 'log' or 'linear'.
%
% "edgelist" is a vector with one more element than "midlist" containing
%   bin edges.
```

6.24 nlProc_getBooleanEdges.m

```
% function [ risesamps fallsamps bothsamps highmidsamps lowmidsamps ] = ...
%   nlFT_getBooleanEdges( boolwave )
%
% This processes a vector of boolean values, and identifies samples that
% are rising edges (first high after a low), falling edges (first low
% after a high), midpoints of high regions, and midpoints of low regions.
%
% The endpoints of the waveform are not considered edges, and the areas
% adjacent to them are not considered to be well-defined high or low regions.
```

```
%
% "boolwave" is a logical vector containing the boolean waveform.
%
% "risesamps" is a vector containing sample indices of rising samples.
% "fallsamps" is a vector containing sample indicies of falling samples.
% "bothsamps" is a vector containing sample indices of both rising and
%   falling samples.
% "highmidsamps" is a vector containing sample indices of the midpoints of
%   regions with high sample values, rounded down.
% "lowmidsamps" is a vector containing sample indices of the midpoints of
%   regions with low sample values, rounded down.
```

6.25 nlProc_getOutlierThresholds.m

```
% function [ threshlow threshhigh midval ] = nlProc_getOutlierThresholds( ...
%   dataseries, lowperc, highperc, lowmult, highmult )
%
% This function computes low and high thresholds for outliers based on
% distance from the median.
%
% This tolerates multidimensional input.
%
% "dataseries" is a vector or matrix containing samples to process.
% "lowperc" is the percentile from which the lower threshold is derived
%   (e.g. 25 for the lower quartile).
% "highperc" is the percentile from which the upper threshold is derived
%   (e.g. 75 for the upper quartile).
% "lowmult" is a multiplier for generating the lower outlier threshold. The
%   distance from the median to the lower percentile value is multiplied by
%   this amount.
% "highmult" is a multiplier for generating the upper outlier threshold. The
%   distance from the median to the upper percentile value is multiplied by
%   this amount.
%
% "threshlow" is the low outlier threshold.
% "threshhigh" is the high outlier threshold.
% "midval" is the median.
```

6.26 nlProc_getOutlierThresholdsQuota.m

```
% function [ threshlow threshhigh midval ] = ...
%   nlProc_getOutlierThresholdsQuota( dataseries, percbase, percmult, ...
%     quotacount )
%
% This function computes low and high threshold for outliers based on
% distance from the median, and adjusts those thresholds to pass at most
```

```
% the requested number of elements.
%
% This tolerates multidimensional input.
%
% "dataseries" is a vector or matrix containing samples to process.
% "percbase" [ low high ] gives the percentiles from which the upper and
%   lower thresholds are derived (e.g. [ 25 75 ] for quartiles ).
% "percmult" [ low high ] gives multipliers for generating outlier thresholds.
%   The distance from the median to the upper/lower percentile value is
%   multiplied by this amount. For quartiles, a multipler of 3 gives 2 sigma.
% "quotacount" [ low high ] gives the maximum number of elements that should
%   be detected past the high or low thresholds. If these are positive
%   values less than 1, they're treated as exponents (i.e. 0.5 means the
%   square root of the total number of elements). Values of NaN ignore quota
%   for the relevant thresholds.
%
% "threshlow" is the low outlier threshold.
% "threshhigh" is the high outlier threshold.
% "midval" is the median.
```

6.27 nlProc_getOutlierTimeRange.m

```
% function [ mintime maxtime threshlow threshhigh threshmedian ] = ...
%   nlProc_getOutlierTimeRange( timeseries, dataseries, ...
%     searchrange, statrange, lowperc, highperc, lowmult, highmult )
%
% This returns the earliest and latest times at which excursions in a signal
% occur. Excursion thresholds are based on distance from the median.
%
% "timeseries" is a vector containing sample timestamps.
% "dataseries" is a vector containing sample data values.
% "searchrange" [ min max ] specifies the time span over which to look for
%   outliers, or [] to search the entire input.
% "statrange" [ min max ] specifies the time span over which to compute
%   median and percentile statistics for thresholding, or [] to compute them
%   over the entire input.
% "lowperc" is the percentile from which the lower threshold is derived
%   (e.g. 25 for the lower quartile).
% "highperc" is the percentile from which the upper threshold is derived
%   (e.g. 75 for the upper quartile).
% "lowmult" is a multiplier for generating the lower outlier threshold. The
%   distance from the median to the lower percentile value is multiplied by
%   this amount.
% "highmult" is a multiplier for generating the upper outlier threshold. The
%   distance from the median to the upper percentile value is multiplied by
%   this amount.
%
% "mintime" is the earliest time at which excursions were detected, or NaN
```

```
%   if no excursions were found.
% "maxtime" is the latest time at which excursions were detected, or NaN if
%   no excursions were found.
% "threshlow" is the low excursion threshold.
% "threshhigh" is the high excursion threshold.
% "threshmedian" is the median value used for computing thresholds.
```

6.28 nlProc_getOutliers.m

```
% function outliervec = nlProc_getOutliers( ...
%   dataseries, lowperc, highperc, lowmult, highmult )
%
% This function flags outliers in a data series based on their distance from
% the median.
%
% This tolerates multidimensional input.
%
% "dataseries" is a vector or matrix containing samples to process.
% "lowperc" is the percentile from which the lower threshold is derived
%   (e.g. 25 for the lower quartile).
% "highperc" is the percentile from which the upper threshold is derived
%   (e.g. 75 for the upper quartile).
% "lowmult" is a multiplier for generating the lower outlier threshold. The
%   distance from the median to the lower percentile value is multiplied by
%   this amount.
% "highmult" is a multiplier for generating the upper outlier threshold. The
%   distance from the median to the upper percentile value is multiplied by
%   this amount.
%
% "outliervec" is a boolean vector or matrix of the same size as
%   "dataseries" that's true for data samples past the outlier thresholds
%   and false otherwise.
```

6.29 nlProc_guessDominantFrequency.m

```
% function [ estfreq estmag ] = nlProc_guessDomiantFrequency( ...
%   wavedata, samprate, freqrange )
%
% This function identifies the highest-magnitude frequency component in the
% supplied waveform and extracts its frequency and magnitude.
%
% "wavedata" is the waveform to analyze.
% "samprate" is the sampling rate.
% "freqrange" [ min max ] is the range of frequencies to consider.
%
% "estfreq" is the estimated frequency of the largest component.
```

```
% "estmag" is the magnitude of that frequency component.
```

6.30 nlProc_guessDominantFrequencyAcrossChans.m

```
% function [ estfreq estmag ] = nlProc_guessDominantFrequencyAcrossChans( ...
%   wavedata, samprate, freqrange, method )
%
% This function identifies the highest-magnitude frequency component within
% a set of supplied waveforms and extracts its frequency and amplitude.
%
% How it does this depends on the "method" argument. For 'average', the
% mean across channels is taken, and that mean signal is analyzed. For
% 'largest', channels are analyzed individually and the response with the
% largest magntiude is chosen. For 'all', channels are analyzed individually
% and the dominant component of each channel is reported.
%
% "wavedata" is a Nchans x Nsamples matrix containing waveform data.
% "samprate" is the sampling rate of the waveform data.
% "freqrange" [ min max ] is the range of frequencies to consider.
% "method" is 'average', 'largest', or 'all'.
%
% "estfreq" is the estimated frequency of the largest component, or a
%   Nchans x 1 vector of frequencies for the 'all' method.
% "extmag" is the magnitude of the largest frequency component, or a
%   Nchans x 1 vector of magnitudes for the 'all' method.
```

6.31 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.
```

6.32 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.
```

6.33 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.
```

6.34 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).
```

6.35 nlProc_interpolateSeries.m

```
% function newdata = nlProc_interpolateSeries( oldtimes, olddata, newtimes )
%
% This performs linear interpolation of a sparsely-defined data series to
% a new set of sparse sample points, handling unusual cases (empty lists,
% NaN entries, etc).
%
% NOTE - To interpolate values off the ends of the source list, the first
% and last points in the source list are used to define a ramp. This gives
% a degraded fit that should still be fairly accurate near the endpoints.
%
% "oldtimes" is a list of time values for which the data series has known
%   values.
```

```
% "olddata" is a list of known data values for these times.
% "newtimes" is a list of time values to produce interpolated data values
%   for.
%
% "newdata" is a list of interpolated data values at the requested times.
```

6.36 nlProc_makeRandomMatrix.m

```
% function newmatrix = nlProc_makeRandomMatrix( ...
%   rowcount, columncount, datarange, rangeexponent )
%
% This fills an N x M matrix with random values drawn from a uniform
% distribution, and then raises the magnitues of these values to the specified
% exponent (preserving sign).
%
% The original uniform distribution has a range chosen such that the
% transformed matrix values span the desired range.
%
% Exponent values greater than 1 tend to produce mostly small values with a
% few large values. Exponent values less than 1 tend to do the opposite.
%
% "rowcount" is the number of rows in the output matrix.
% "columncount" is the number of columns in the output matrix.
% "datarange" [ min max ] is the desired range of output values.
% "rangeexponent" is the power to raise magnitudes to.
%
% "newmatrix" is a N x M matrix with cell values spanning the specified range.
```

6.37 nlProc_mergeNearbyValues.m

```
% function [ newlist newidxfromold oldidxfromnew ] = ...
%   nlProc_mergeNearbyValues( oldlist, winsize )
%
% This function merges nearby entries in a list of values, returning a
% smaller list and a lookup table indicating which new value corresponds
% to each old list entry.
%
% FIXME - This uses an O(n2) algorithm!
%
% "oldlist" is a vector containing values to group.
% "winsize" is a scalar indicating the window size for grouping.
%
% "newlist" is a vector containing a sorted list of new values.
% "newidxfromold" is a vector of the same size as "oldlist" that contains the
%   location in "newlist" corresponding to each entry in "oldlist".
% "oldidxfromnew" is a cell array of the same size as "newlist" that contains
```

```
%   vectors holding all locations in "oldlist" corresponding values in
%   "newlist".
```

6.38 nlProc_normalizeMatrixColumns.m

```
% function newmatrix = nlProc_normalizeMatrixColumns( oldmatrix, normrange )
%
% This normalizes the columns of a matrix so that for each column the sum of
% that column's absolute values is equal to a column quota value drawn from
% the specified normalization range.
%
% This works most intuitively with a data range of [ 0 1 ], but can work
% with any real-valued data.
%
% "oldmatrix" is the matrix to normalize.
% "normrange" [ min max ] is the range of values to draw each column's quota
%   from. Quota values are uniformly distributed in this range, and each
%   column is normalized so that its absolute values sum to the quota.
%
% "newmatrix" is a copy of "oldmatrix" with columns normalized.
```

6.39 nlProc_normalizeMatrixRows.m

```
% function newmatrix = nlProc_normalizeMatrixRows( oldmatrix, normrange )
%
% This normalizes the rows of a matrix so that for each row the sum of that
% row's absolute values is equal to a row quota value drawn from the
% specified normalization range.
%
% This works most intuitively with a data range of [ 0 1 ], but can work
% with any real-valued data.
%
% "oldmatrix" is the matrix to normalize.
% "normrange" [ min max ] is the range of values to draw each row's quota
%   from. Quota values are uniformly distributed in this range, and each row
%   is normalized so that its absolute values sum to the quota.
%
% "newmatrix" is a copy of "oldmatrix" with rows normalized.
```

6.40 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.
```

6.41 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.
```

6.42 nlProc_padHeatmapGaps.m

```
% function [ newdata newxseries newyseries ] = ...
%   nlProc_padHeatmapGaps( olddata, oldxseries, oldyseries )
%
% This looks for nonuniformities in row and column coordinates in a
% supplied matrix of two-dimensional data, and pads any sufficiently large
% gaps with new cells containing NaN.
%
% The intention is to simplify plotting heatmaps and histograms of data
```

```
% with independent axes that have discontinuous ranges.
%
% To produce sensible output, oldxseries and oldyseries should mostly
% contain linearly spaced values. The "nominal" spacing is computed as the
% median of the difference between successive values.
%
% "olddata" is a matrix indexed by (y,x) that contains data values.
% "oldxseries" is a vector containing X axis values for each column in the
%   data matrix.
% "oldyseries" is a vector containing Y axis values for each row in the
%   data matrix.
%
% "newdata" is a copy of "olddata" that may have additional rows and columns
%   added which contain NaN values.
% "newxseries" is a copy of "oldxseries" that may have additional values.
% "newyseries" is a copy of "oldyseries" that may have additional values.
```

6.43 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.
```

6.44 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.
```

6.45 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.
```

6.46 nlProc_sampleWaveAtTimes.m

```
% function query_vals = ...
%   nlProc_sampleWaveAtTimes( signal_times, signal_vals, query_times )
%
% This function measures the value of a signal waveform at a list of
% specified times.
%
% The signal is assumed to be uniformly sampled ("signal_times" is a linear
% sequence). The signal must contain at least two samples.
%
% "signal_times" is a vector containing waveform timestamps. This should be
%   a linear sequence (uniform sampling).
% "signal_vals" is a vector containing waveform values at each timestamp.
% "query_times" is a vector containing times at which to measure the
%   signal waveform. There are no constraints on these values.
%
% "query_vals" is a vector containing waveform values at the times listed
%   in "query_times". Out-of-range timestamps get NaN values.
```

6.47 nlProc_squashMatrixDiagonal.m

```
% function newmatrix = nlProc_squashMatrixDiagonal( oldmatrix )
%
% This sets all diagonal elements of the input matrix to zero.
```

```
% The input matrix does not have to be square.
%
% "oldmatrix" is the matrix to modify.
%
% "newmatrix" is a copy of "oldmatrix" with diagonal elements set to zero.
```

6.48 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.
```

6.49 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.
```

6.50 nlProc_trimTimeSequence.m

```
% function newlist = nlProc_trimTimeSequence( oldlist, timespan )
%
% This accepts a list of timestamps and removes any that are outside of the
% specified timespan.
%
% "oldlist" is a vector of timestamps to prune.
```

```
% "timespan" [ min max ] specifies the range of accepted timestamps.
%
% "newlist" is a vector of timestamps that were within the desired
% timestamp range.
```

Chapter 7

"nlSynth" Functions

7.1 nlSynth_ftWrapper_robinsonSimulateHindriksNetwork.m

```
% function ftdata = nlSynth_ftWrapper_robinsonSimulateHindriksNetwork( ...
%   trialcount, triggertime, poplabels, wantprogress, ...
%   duration, startup, timestep, modelparams, intcouplings, ...
%   popcount, cortexmixing, cortexdelays_ms )
%
% This is a wrapper for nlSynth_robinsonSimulateHindriksNetwork().
% See that function's documentation for details.
%
% This simulates cortex and thalamus neural activity, using the model from
% Robinson 2002 with augmented input per Freyer 2011 and Hindriks 2023:
%
% https://journals.aps.org/pre/abstract/10.1103/PhysRevE.65.041924
% https://www.jneurosci.org/content/31/17/6353.short
% https://www.nature.com/articles/s42003-023-04648-x
%
% "trialcount" is the number of Field Trip trials to simulate.
% "triggertime" is the trigger offset from the start of simulation (seconds).
% "poplabels" is a cell array containing Field Trip channel names for the
%   neural populations, or {} to automatically generate channel names.
% "wantprogress" is true to write progress messages to the console, false
%   otherwise.
%
% Remaining arguments are per nlSynth_robinsonSimulateHindriksNetwork().
%
% "ftdata" is a ft_datatype_raw structure containing trial data, including
%   a header (hdr) and a config structure with trial definitions (cfg.trl).
```

7.2 nlSynth_origHindriksGenerateRobinsonNoise.m

```
% function timeseries = nlSynth_origHindriksGenerateRobinsonNoise( ...
%   noises,coupling,h,T,NN,sc_matrix,delay_matrix,Ksi)
%
% Renamed from Robinson_state_dependent_noise().
% This is the original code, copied and re-licensed (with permission) from:
% https://github.com/Prejaas/amplitudecoupling
% And published in:
% https://www.nature.com/articles/s42003-023-04648-x
%
% Original comments follow:
%
% Hindriks/Tewarie 2022 simulate Robinson model with state dependent noise
% Dissociation between phase and power correlation networks in
% the human brain can be explained by co-occurrent bursts
%
% input parameters:
% noises        - mean noise level
% coupling      - coupling parameter between two populations
% h             - integration time step
% T             - simulation time in seconds
% NN            - number of populations or nodes
% sc_matrix     - structural connectivity matrix (NN x NN)
% delay_matrix  - include heterogenous delays (NN x NN)
% Ksi           - state dependent noise parameter
%
% output parameters
% timeseries    - simulated timeseries (firing rate cortical excitatory
% population)
```

7.3 nlSynth_robinsonAddLoopGainInfo.m

```
% function newloopinfo = nlSynth_robinsonAddLoopGainInfo( ...
%   oldloopinfo, edgegains, edgegaingradients )
%
% This accepts a loop metadata structure array and augments each record
% with gain-related information.
%
% An empty cell array may be supplied for the gain gradients, to omit
% gradient information.
%
% "oldloopinfo" is a structure returned by nlSynth_robinsonFindLoops().
% "edgegains" is a 4x4 matrix indexed by (destination, source) that contains
%   the small-signal firing rate gains between each source and destination
%   for the excitatory, inhibitory, specific nucleus, and reticular nucleus
%   neural populations.
```

```
% "edgegaingradients" is a 4x4 cell array indexed by (destination, source)
%   that contains the gradient with respect to "intcouplings" of the
%   small-signal firing rate gains in "edgegains". Passing an empty cell
%   array skips calculation of loop gain gradients.
%
% "newloopinfo" is a copy of "oldloopinfo" with the following fields added
%   to each record:
%   "cyclegainraw" is the small-signal gain from traversing once around the
%     loop, without taking into account filter attenuation.
%   "cyclegain" is the small-signal gain from traversing once around the
%     loop with filter attenuation taken into account.
%   "envelopetau" is the time constant for the growth (positive) or decay
%     (negative) of the oscillation envelope. The envelope is exp(t/tau).
%   "cyclegaingradient" is the gradient with respect to "intcouplings" of
%     "cyclegain". This is a 4x4 matrix (per "intcouplings"). If an empty
%     cell array is passed as "edgegaingradients", this field is omitted.
```

7.4 nlSynth_robinsonEstimateOperatingPointExponential.m

```
% function [ firingrates potentials ] = ...
%   nlSynth_robinsonEstimateOperatingPointExponential( ...
%     modelparams, intcouplings, startpotentials )
%
% This attempts to estimate the DC operating point of a Robinson neural model.
%
% Per the model guide, operating points with firing rates much less than the
% maximum are solutions to the equation:
%
% potentials = intcouplings * Q_0 * exp( potentials / sigmaprime )
%
% This function does a brute-force gradient descent search for operating
% points using "fsolve". This only finds one point; several points may
% exist.
%
% NOTE - Operating point firing rates _must_ be examined to confirm that
% they are much less than modelparams.qmax. If they are not several times
% smaller, the operating point is not correct.
%
% "modelparams" is a model parameter structure with the fields described in
%   MODELPARAMSROBINSON.txt.
% "intcouplings" is a 4x4 matrix indexed by (destination,source) that
%   provides the coupling weights (in mV*s) between excitatory, inhibitory,
%   specific nucleus, and reticular nucleus neural populations.
% "startpotentials" is a vector containing cell potentials for the excitatory,
%   inhibitory, specific nucleus, and reticular nucleus populations used as
%   a starting point for further optimization. Set this to [] to call
%   nlSynth_robinsonEstimateOperatingPointLinear() to generate starting
%   potentials.
```

```
%
% "firingrates" is a vector containing firing rates for the excitatory,
%   inhibitory, specific nucleus, and reticular nucleus populations.
% "potentials" is a vector containing cell potentials for the excitatory,
%   inhibitory, specific nucleus, and reticular nucleus populations.
```

7.5 nlSynth_robinsonEstimateOperatingPointLinear.m

```
% function [ firingrates potentials ] = ...
%   nlSynth_robinsonEstimateOperatingPointLinear( modelparams, intcouplings )
%
% This attempts to estimate the DC operating point of a Robinson neural model.
%
% Per the model guide, operating points with firing rates much less than the
% maximum are solutions to the equation:
%
% potentials = intcouplings * Q_0 * exp( potentials / sigmaprime )
%
% This function uses a linear approximation to exp(x) to estimate operating
% points for potentials that are small compared to sigmaprime. NOTE - This
% is not a robus assumption! The operating point _must_ be examined to
% confirm that this condition holds. If it doesn't hold, the estimated
% operating point is not correct.
%
% "modelparams" is a model parameter structure with the fields described in
%   MODELPARAMSROBINSON.txt.
% "intcouplings" is a 4x4 matrix indexed by (destination,source) that
%   provides the coupling weights (in mV*s) between excitatory, inhibitory,
%   specific nucleus, and reticular nucleus neural populations.
%
% "firingrates" is a vector containing firing rates for the excitatory,
%   inhibitory, specific nucleus, and reticular nucleus populations.
% "potentials" is a vector containing cell potentials for the excitatory,
%   inhibitory, specific nucleus, and reticular nucleus populations.
```

7.6 nlSynth_robinsonFindLoops.m

```
% function loopinfo = ...
%   nlSynth_robinsonFindLoops( modelparams, intcouplings, minweight )
%
% This examines a Robinson model coupling matrix and identifies loops.
% Loop metadata is extracted.
%
% This does not extract loop gain, since that varies with operating point.
%
% "modelparams" is a model parameter structure with the fields described in
```

```
%   MODELPARAMSROBINSON.txt.
% "intcouplings" is a 4x4 matrix indexed by (destination, source) that
%   provides the coupling weights (in mV*s) between excitatory, inhibitory,
%   specific nucleus, and reticular nucleus neural populations.
% "minweight" is the threshold to use when evaluating whether a coupling
%   weight is nonzero (the absolute value must be at least "minweight").
%
% "loopinfo" is a structure array with one element per identified loop, and
%   the following fields:
%   "label" is a unique identifier for the loop (the concatenation of the
%     letters associated with each region in the loop's path).
%   "regionsvisited" is a vector containing the indices of each region in the
%     loop's path. The first index is repeated as the last index.
%   "delay" is the propagation time for one cycle around the loop, in seconds.
%   "isinverting" is true if the product of the loop's edge couplings is
%     negative, and false if the product is positive.
%   "frequency" is the loop's fundamental mode frequency in Hz (1/delay if
%     non-inverting, half that if inverting).
%   "attenuation" is the loop's attenuation at its fundamental mode
%     frequency from the alpha, beta, and gamma model parameters. This will
%     be between 1 if the signal is passed perfectly and less than 1 if not.
```

7.7 nlSynth_robinsonGetEdgeGainGradients.m

```
% function edgegaingradients = nlSynth_robinsonGetEdgeGainGradients( ...
%   modelparams, intcouplings, firingrates, rategradients )
%
% This function calculates the gradient of the small-signal gains of each
% network edge in a Robinson neural model with respect to the internal
% coupling matrix, at a specified operating point.
%
% This assumes firing rates that are much less than the maximum rate, and
% assumes that the gradients of the firing rates are already known.
%
% "modelparams" is a model parameter structure with the fields described in
%   MODELPARAMSROBINSON.txt.
% "intcouplings" is a 4x4 matrix indexed by (destination, source) that
%   provides the coupling weights (in mV*s) between excitatory, inhibitory,
%   specific nucleus, and reticular nucleus neural populations.
% "firingrates" is a vector specifying the operating point firing rates of
%   excitatory, inhibitory, specific nucleus, and reticular nucleus neural
%   populations.
% "rategradients" is a cell array with 4 cells, containing gradient matrices
%   with respect to "intcouplings" for the excitatory, inhibitory, specific
%   nucleus, and reticular nucleus firing rates.
%
% "edgegaingradients" is a 4x4 cell array indexed by (destination, source)
%   that contains the gradient with respect to "intcouplings" of the
```

```
%   small-signal firing rate gains between each source and destination for
%   the excitatory, inhibitory, specific nucleus, and reticular nucleus
%   neural populations.
```

7.8 nlSynth_robinsonGetEdgeGains.m

```
% function edgegains = nlSynth_robinsonGetEdgeGains( ...
%   modelparams, intcouplings, firingrates )
%
% This function estimates the small-signal gain of each network edge in a
% Robinson neural model, at a specified operating point.
%
% "modelparams" is a model parameter structure with the fields described in
%   MODELPARAMSROBINSON.txt.
% "intcouplings" is a 4x4 matrix indexed by (destination, source) that
%   provides the coupling weights (in mV*s) between excitatory, inhibitory,
%   specific nucleus, and reticular nucleus neural populations.
% "firingrates" is a vector specifying the operating point firing rates of
%   excitatory, inhibitory, specific nucleus, and reticular nucleus neural
%   populations.
%
% "edgegains" is a 4x4 matrix indexed by (destination, source) that contains
%   the small-signal firing rate gains between each source and destination
%   for the excitatory, inhibitory, specific nucleus, and reticular nucleus
%   neural populations.
```

7.9 nlSynth_robinsonGetModelParamsFreyer.m

```
% function [ modelparams intcouplings ] = ...
%   nlSynth_robinsonGetModelParamsFreyer()
%
% This returns model and coupling parameters for use with
% nlSynth_robinsonStepCortexThalamus() and related functions.
%
% Values are the ones used in Freyer 2011 (Table 1):
% https://www.jneurosci.org/content/31/17/6353.short
%
% These have slightly adjusted alpha and beta time constants and
% substantially different coupling weights vs Robinson 2002.
%
% No arguments.
%
% "modelparams" is a model parameter structure with the fields described
%   in MODELPARAMSROBINSON.txt.
% "intcouplings" is a 4x4 matrix indexed by (destination,source) that
%   provides the coupling weights (in mV*s) between excitatory, inhibitory,
```

```
%   specific nucleus, and reticular nucleus neurons.
```

7.10 nlSynth_robinsonGetModelParamsHindriks.m

```
% function [ modelparams intcouplings ] = ...
%   nlSynth_robinsonGetModelParamsHindriks()
%
% This returns model and coupling parameters for use with
% nlSynth_robinsonStepCortexThalamus() and related functions.
%
% Values are the ones used in Hindriks 2023:
% https://www.nature.com/articles/s42003-023-04648-x
% https://github.com/Prejaas/amplitudecoupling
%
% These are identical to the Robinson 2002 values, except with a smaller
% noise coupling coefficient.
%
% No arguments.
%
% "modelparams" is a model parameter structure with the fields described
%   in MODELPARAMSROBINSON.txt.
% "intcouplings" is a 4x4 matrix indexed by (destination,source) that
%   provides the coupling weights (in mV*s) between excitatory, inhibitory,
%   specific nucleus, and reticular nucleus neurons.
```

7.11 nlSynth_robinsonGetModelParamsRobinson.m

```
% function [ modelparams intcouplings ] = ...
%   nlSynth_robinsonGetModelParamsRobinson()
%
% This returns model and coupling parameters for use with
% nlSynth_robinsonStepCortexThalamus() and related functions.
%
% Values are the ones used in Robinson 2002 (Table 1):
% https://journals.aps.org/pre/abstract/10.1103/PhysRevE.65.041924
%
% No arguments.
%
% "modelparams" is a model parameter structure with the fields described
%   in MODELPARAMSROBINSON.txt.
% "intcouplings" is a 4x4 matrix indexed by (destination,source) that
%   provides the coupling weights (in mV*s) between excitatory, inhibitory,
%   specific nucleus, and reticular nucleus neurons.
```

7.12 nlSynth_robinsonGetOperatingPointGradient.m

```
% function [ rategradients potentialgradients ] = ...
%   nlSynth_robinsonGetOperatingPointGradient( ...
%     modelparams, intcouplings, testpotentials, couplingstep, zerohandling )
%
% This function attempts to estimate the gradient of the DC operating point
% of a Robinson neural model with respect to the internal coupling matrix.
%
% This assumes firing rates that are much less than the maximum rate. Per the
% model guide, operating points under these conditions are solutions to:
%
% potentials = intcouplings * Q_0 * exp( potentials / sigmaprime )
%
% The gradient of this function is evaluated numerically, by perturbing the
% coupling matrix and finding operating points for each perturbed version.
%
% "modelparams" is a model parameter structure with the fields described in
%   MODELPARAMSROBINSON.txt.
% "intcouplings" is a 4x4 matrix indexed by (destination, source) that
%   provides the coupling weights (in mV*s) between excitatory, inhibitory,
%   specific nucleus, and reticular nucleus neural populations.
% "testpotentials" is a vector containing cell potentials for the excitatory,
%   inhibitory, specific nucleus, and reticular nucleus populations at the
%   test point to analyze. This is used as the starting point for the
%   operating point search.
% "couplingstep" is a scalar indicating the amount by which each coupling
%   value should be perturbed when evaluating the gradient numerically. This
%   should be much smaller than the magnitude of nonzero coupling values.
% "zerohandling" is 'all' to compute the gradient with respect to all
%   coupling values, and 'nonzero' to compute the gradient with respect to
%   all coupling values with magnitudes larger than "couplingstep" (storing
%   NaN as the gradient for coupling values that are smaller than this).
%
% "rategradients" is a cell array with 4 cells, containing gradient matrices
%   with respect to "intcouplings" for the excitatory, inhibitory, specific
%   nucleus, and reticular nucleus firing rates.
% "potentialgradients" is a cell array with 4 cells, containing gradient
%   matrices with respect to "intcouplings" for the excitatory, inhibitory,
%   specific nucleus, and reticular nucleus cell potentials.
```

7.13 nlSynth_robinsonGetRegionInfo.m

```
% function [ indices_lut names_lut ]  = nlSynth_robinsonGetRegionInfo()
%
% This returns metadata associating each region simulated by the Robinson
% 2002 model with a row/column index, a pretty name, and abbreviated names.
```

```
%
% No arguments.
%
% "indices_lut" is a structure with the following fields:
%   "cortex_excitatory" is the row/column index corresponding to excitatory
%     neurons in the cortex.
%   "cortex_inhibitory" is the row/column index corresponding to inhibitory
%     neurons in the cortex.
%   "thalamus_specific" is the row/column index corresponding to "specific
%     nucleus" neurons in the thalamus (also called the relay population).
%   "thalamus_reticular" is the row/column index corresponding to "reticular
%     nucleus" neurons in the thalamus.
%
% "names_lut" is a structure array indexed by region number, with the
%   following fields (all character vectors):
%   "title" is a verbose plot-safe name.
%   "label" is a terse filename-safe and plot-safe name.
%   "letter" is a capital letter.
%   "lutfield" is the name of the corresponding field in "indices_lut".
```

7.14 nlSynth_robinsonGetSigmoid.m

```
% function firingrate = nlSynth_robinsonGetSigmoid( ...
%   potential, maxrate, threshlevel, threshdeviation )
%
% This converts a cell-body potential (V in Robinson's model) to a firing
% rate (Q in Robinson's model).
%
% This is described in eq. 1 of Robinson 2002:
% https://journals.aps.org/pre/abstract/10.1103/PhysRevE.65.041924
% And in terms of sigma, not sigma-prime, in eq. m4 of Freyer 2011:
% https://www.jneurosci.org/content/31/17/6353.short
%
% "potential" is the cell body potential (V) to convert. This may be a
%   vector or matrix, to convert several potentials.
% "maxrate" is the maximum firing rate (Q_max).
% "threshlevel" is the average neuron threshold (theta).
% "threshdeviation" is the standard deviation of the average neuron threshold
%   (sigma - not sigma prime!).
%
% "firingrate" is the resulting firing rate (Q, or S(V)).
```

7.15 nlSynth_robinsonGetSigmoidDerivative.m

```
% function dQdV = nlSynth_robinsonGetSigmoidDerivative( ...
%   potential, maxrate, threshlevel, threshdeviation )
```

```
%
% This gets the derivative of the Robinson 2002 activation function with
% respect to cell-body potential, for a given potential.
%
% "potential" is the cell body potential (V) to evaluate the derivative at.
%   This may be a vector or matrix, to evaluate several potentials.
% "maxrate" is the maximum firing rate (Q_max).
% "threshlevel" is the average neuron threshold (theta).
% "threshdeviation" is the standard deviation of the average neuron threshold
%   (sigma - not sigma prime!).
%
% "dQdV" is the derivative of the firing rate with respect to potential,
%   at the specified potential.
```

7.16 nlSynth_robinsonGetSigmoidInverse.m

```
% function potential = nlSynth_robinsonGetSigmoidInverse( ...
%   firingrate, maxrate, threshlevel, threshdeviation )
%
% This converts a firing rate (Q in Robinson's model) back to a cell-body
% potential (V in Robinson's model), performing the inverse of
% nlSynth_robinsonGetSigmoid().
%
% "firingrate" is the firing rate (Q, or S(V)). This may be a vector or
%   matrix, to convert several firing rates into potentials.
% "maxrate" is the maximum firing rate (Q_max).
% "threshlevel" is the average neuron threshold (theta).
% "threshdeviation" is the standard deviation of the average neuron threshold
%   (sigma - not sigma prime!).
%
% "potential" is the cell body potential (V).
```

7.17 nlSynth_robinsonSimulateHindriksNetwork.m

```
% function firingrates = nlSynth_robinsonSimulateHindriksNetwork( ...
%   duration, startup, timestep, modelparams, intcouplings, ...
%   popcount, cortexmixing, cortexdelays_ms )
%
% This simulates cortex and thalamus neural activity, using the model from
% Robinson 2002 with augmented input per Freyer 2011 and Hindriks 2023:
%
% https://journals.aps.org/pre/abstract/10.1103/PhysRevE.65.041924
% https://www.jneurosci.org/content/31/17/6353.short
% https://www.nature.com/articles/s42003-023-04648-x
%
% This simulates N populations of excitatory and inhibitory neurons in the
```

```
% cortex, and N populations of neurons in the specific and reticular nuclei
% in the thalamus (per Freyer 2011). Excitatory neuron populations in the
% cortex interact with each other, per Hindriks 2023.
%
% NOTE - This uses the forward Euler method for evolving system state.
% This is numerically stable if and only if the time step is much smaller
% than the time scales of any system dynamics. Make this much smaller than
% you think you need to.
%
% NOTE - If multiple durations are specifies, various parameters _may_ be
% specified for each epoch rather than globally. This is optional; any
% parameters that are not specified per-epoch are duplicated as needed.
%
% "duration" is the number of seconds to simulate. NOTE - This may be a
%   vector, specifying several successive durations which may have
%   different simulation parameters.
% "startup" is the number of seconds to simulate before the duration to
%   allow the simulation to settle/converge. This is typically 2-5 seconds.
% "timestep" is the amount of time to advance the simulation during each
%   sample, in seconds. NOTE - This must be much smaller than system
%   dynamics timescales!
% "modelparams" is a structure specifying model tuning parameters, per
%   MODELPARAMSROBINSON.txt. NOTE - If "duration" is a vector, this may
%   optionally be a struct array specifying different parameters for each
%   epoch.
% "intcouplings" is a 4x4 matrix indexed by (destination,source) that
%   provides the coupling weights (in mV*s) between excitatory cortex
%   neurons (1), inhibitory cortex neurons (2), specific nucleus neurons (3),
%   and reticular neurons (4). Typical coupling range from -2 to +2.
%   NOTE - If "duration" is a vector, this may optionally be a 4x4xN matrix
%   specifying different couplings for each epoch.
% "popcount" is the number of neural populations to simulate (Npop).
% "cortexmixing" is a Npop x Npop matrix indexed by (destination,source)
%   that specifies the internal communication mixing of excitatory neurons
%   in the cortex. After mixing, these then get weighted by (scalar)
%   modelparams.mixturecoupling before integration as neural inputs. Set
%   this to [] to omit internal cortex communication/mixing.
%   NOTE - The typical mixture coupling weight in MODELPARAMSROBINSON.txt
%   assumes that the cortex mixing matrix has rows normalized so that the
%   absolute values of each row's elements sums to 1 (or a value close to 1).
%   NOTE - If "duration" is a vector, this may optionally be a Npop x Npop x N
%   matrix specifying different mixing weights for each epoch.
% "cortexdelays_ms" is a Npop x Npop matrix indexed by (destination,source)
%   that specifies the internal communication delays of excitatory neurons
%   in the cortex, in milliseconds. Set this to [] to omit internal cortex
%   communication/mixing.
%   NOTE - If "duration" is a vector, this may optionally be a Npop x Npop x N
%   matrix specifying different delays for each epoch.
%
% "firingrates" is a 4 x Npop x Nsamples matrix containing firing rates
```

```
%   for excitatory cortex neurons (1), inhibitory cortex neurons (2),
%   specific nucleus neurons (3), and reticular neurons (4). The excitatory
%   cortex neuron firing rates are gamma-damped, per Robinson 2002.
```

7.18 nlSynth_robinsonStepCortexThalamus.m

```
% function statefuture = nlSynth_robinsonStepCortexThalamus( modelparams, ...
%   timestep, statepresent, statepast, intcouplings, extrates, extcouplings )
%
% This generates the future state of a cortico-thalamic loop given the
% present state, using the model from Robinson 2002 with augmented input
% per Freyer 2011 and Hindriks 2023:
%
% https://journals.aps.org/pre/abstract/10.1103/PhysRevE.65.041924
% https://www.jneurosci.org/content/31/17/6353.short
% https://www.nature.com/articles/s42003-023-04648-x
%
% This simulates N populations of excitatory and inhibatory neurons in the
% cortex, and N populations of neurons in the specific and reticular nuclei
% in the thalamus (per Freyer 2011). Population bins are independent of
% each other (communication is handled by the caller).
%
% NOTE - This uses the forward Euler method for evolving system state.
% This is numerically stable if and only if the time step is much smaller
% than the time scales of any system dynamics. Make this much smaller than
% you think you need to.
%
% "modelparams" is a structure with the following fields (as described in
%   MODELPARAMSROBINSON.txt):
%   "qmax" is the maximum firing rate (1/sec); typically 250.
%   "threshlevel" is the average neuronal threshold (mV); typ. 15.
%   "threshsigma" is the standard deviation of the neuronal threshold (mV);
%     typically 6.0.
%   "alpha" is the inverse decay time (1/sec); typically 50.
%   "beta" is the inverse rise time (1/sec); typically 200.
%   "gamma" is the inverse within-cortex propagation time (1/sec); typ. 100.
% "timestep" is the amount of time to advance the simulation by (in seconds).
%   NOTE - This must be much smaller than system dynamics timescales!
% "statepresent" is a structure with the following fields:
%   "potentials" is a 4xN matrix containing Ve_k, Vi_k, Vs_k, and Vr_k from
%     the Robinson model.
%   "velocities" is a 4xN matrix containing the approximate first time
%     derivative of "potentials".
%   "cortexrates" is a 1xN matrix containing the gamma-damped firing rate
%     of cortex excitatory neurons (phi_e).
%   "cortexvelocities" is a 1xN matrix containing the approxmate first time
%     derivative of "cortexrates".
% "statepast" is a structure with the same fields as "statepresent", taken
```

```
%   from a past stimulation step. This should be delayed by the one-way
%   cortex/thalamus communication time (one half of the round-trip delay).
% "intcouplings" is a 4x4 matrix indexed by (destination,source) that
%   provides the coupling weights (in mV*s) between excitatory cortex
%   neurons (1), inhibitory cortex neurons (2), specific nucleus neurons (3),
%   and reticular nucleus neurons (4). Typical couplings range from -2 to +2.
% "extrates" is a MxN matrix containing the firing rates of M external inputs
%   to the system. These may represent noise (as with Freyer 2011) or
%   cross-coupled activity within the cortex (as with Hindriks 2023).
%   This may be empty (M = 0).
% "extcouplings" is a 4xM matrix indexed by (destination,source) that
%   provides the coupling weights (in mV*s) between the internal model
%   neurons (destinations) and the system's external inputs (sources).
%   This may be empty (M = 0).
%
% "statefuture" is a copy of "statepresent" advanced by one time step.
```

Chapter 8

"nlUtil" Functions

8.1 nlUtil_concatStructArrays.m

```
% function newlist = nlUtil_concatStructArrays(firstlist, secondlist)
%
% This concatenates two structure arrays. It does this field by field, so
% that any missing fields are added and initialized.
%
% If the two source structures are known to have the same fields, use
% horzcat/vertcat instead.
%
% "firstlist" is a structure array.
% "secondlist" is a structure array.
%
% "newlist" is a structure array containing the elements of "firstlist" and
%   "secondlist".
```

8.2 nlUtil_confirmStructureFields.m

```
% function [ newlist passvec ] = ...
%   nlUtil_confirmStructureFields( oldlist, fieldswanted )
%
% This accepts a structure array or a cell array of structures and builds a
% new list containing only those records that include the specified fields.
%
% "oldlist" is a structure array or a cell array containing structures.
% "fieldswanted" is a cell array containing the names of fields that must be
%   present.
%
% "newlist" is a copy of "oldlist" containing only those records that have
%   the desired fields.
% "passvec" is a logical vector such that newlist = oldlist(passvec).
```

8.3 nlUtil_continuousToSparse.m

```
% function [ eventvals, eventtimes ] = ...
%   nlUtil_continuousToSparse( wavedata, timedata )
%
% This converts a continuous discrete-valued waveform into a series of
% nonuniformly-sampled events (representing changes in the waveform's value).
%
% The waveform value is assumed to be discrete and changes are assumed to be
% infrequent. This will still work if given floating-point data or data that
% changes at every sample, but there's little point in representing these
% signals as event lists ("eventvals" and "eventtimes" will be copies of
% "wavedata" and "timedata", respectively).
%
% NOTE - There will always be an event at the first timestamp representing
% the initial data value.
%
% "wavedata" is a vector with samples from the continuous signal.
% "timedata" is a vector of timestamps corresponding to these samples.
%
% "eventvals" is a vector containing signal values immediately after changes.
% "eventtimes" is a vector containing the timestamps of these changes.
```

8.4 nlUtil_extractStructureSeries.m

```
% function dataseries = ...
%   nlUtil_extractStructureSeries( structlist, fieldwanted )
%
% This accepts a structure array or a cell array of structures and compiles
% a data series containing values extracted from a specific structure field.
%
% This is intended for computing global statistics across data series stored
% in records.
%
% This will tolerate non-numeric data (such as cell arrays).
%
% If the requested field isn't found, an empty vector will be returned.
%
% "structlist" is a structure array or a cell array containing structures.
% "fieldwanted" is the name of the field to extract.
%
% "dataseries" is a 1xN vector containing the concatenated data series from
%   the specified structure fields. Each field's contents is reshaped to
%   a linear vector before being concatenated.
```

8.5 nlUtil_findXMLStructNodesRecursing.m

```
% function matches = nlUtil_findXMLStructNodesRecursing( ...
%   xmlstruct, tagswanted, attribswanted )
%
% This searches through an XML parse structure (returned by readstruct()),
% and identifies tree nodes at any level that match desired criteria. This
% recurses within non-matching nodes (but not matching nodes).
%
% This will not match the top-level structure itself (if given an empty tags
% list and a non-empty attributes list). To test against that case, pass
% "struct( 'toplevel', xmlstruct )" as the input parse tree.
%
% "xmlstruct" is a structure containing an XML parse tree (per readstruct()).
% "tagswanted" is a cell array containing the names of tags to match. If this
%   is empty, all nodes match. Matching is not sensitive to case.
% "attribswanted" is a cell array containing names and values of attributes
%   to match. If _all_ specified attributes match _any_ of their permitted
%   values, or if "attribswanted" is empty, the node matches. The cell array
%   has the form { 'attr1', { 'val1', 'val2' }, 'attr2', ... }. Matched
%   values are character arrays and are not sensitive to case.
%
% "matches" is a cell array containing parse tree nodes that matched the
%   selection criteria. These are themselves XML parse structures.
```

8.6 nlUtil_findXMLStructNodesTopLevel.m

```
% function [ matches nonmatches ] = nlUtil_findXMLStructNodesTopLevel( ...
%   xmlstruct, tagswanted, attribswanted )
%
% This searches through an XML parse structure (returned by readstruct()),
% and identifies tree nodes at the top level that match desired criteria.
%
% This will not match the top-level structure itself (if given an empty tags
% list and a non-empty attributes list). To test against that case, pass
% "struct( 'toplevel', xmlstruct )" as the input parse tree.
%
% "xmlstruct" is a structure containing an XML parse tree (per readstruct()).
% "tagswanted" is a cell array containing the names of tags to match. If this
%   is empty, all nodes match. Matching is not sensitive to case.
% "attribswanted" is a cell array containing names and values of attributes
%   to match. If _all_ specified attributes match _any_ of their permitted
%   values, or if "attribswanted" is empty, the node matches. The cell array
%   has the form { 'attr1', { 'val1', 'val2' }, 'attr2', ... }. Matched
%   values are character arrays and are not sensitive to case.
%
% "matches" is a cell array containing top-level nodes that matched the
```

```
%   selection criteria. These are themselves XML parse structures.
% "nonmatches" is a cell array containing top-level nodes that did not match
%   the selection criteria. These are themselves XML parse structures.
```

8.7 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.
```

8.8 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.
```

8.9 nlUtil_getCellOfStructField.m

```
% function resultlist = nlUtil_getCellOfStructField( ...
%   cellofstruct, desiredfield, defaultvalue )
%
% This accepts a cell array of structures, examines each structure for the
% specified field, and returns a cell array or vector containing the field
% contents from each structure.
%
% This is intended to do what "{ foostruct.fieldname }" and
% "[ foostruct.fieldname ]" do for struct arrays, with cell arrays of
% structures (used when field names aren't consistent).
%
% "cellofstruct" is a cell array, where each cell contains a struct or a
%   struct array.
% "desiredfield" is the name of the field to look for.
% "defaultvalue" is the value to return for structures that are missing the
%   desired field. This also indicates the data type to look for; if
%   "defaultvalue" is a character vector, fields are assumed to contain
```

```
%   character vectors; otherwise they're assumed to have numeric or boolean
%   content.
%
% "resultlist" is a 1xN cell array (if handling character data) or vector
%   (if handling numeric/boolean data) with one entry per struct encountered
%   in the input.
```

8.10 nlUtil_getCommonTimeRanges.m

```
% function [ firstsampmask secondsampmask ] = ...
%   nlUtil_getCommonTimeRanges( firsttimes, secondtimes )
%
% This accepts two ascending-sorted series and finds the span of samples
% within each that cover overlapping value ranges.
%
% This is mostly used to find time-aligned subsets of two series using
% timestamp values.
%
% "firsttimes" is a vector of timestamp values in ascending order.
% "secondtimes" is a vector of timestamp values in ascending order.
%
% "firstsampmask" is a mask vector tha's true for samples in "firsttimes"
%   that have values within the minimum and maximum range of both input series.
% "secondsampmask" is a mask vector tha's true for samples in "secondtimes"
%   that have values within the minimum and maximum range of both input series.
```

8.11 nlUtil_getLabelIndices.m

```
% function indexlist = nlUtil_getLabelIndices( labellist, labellut )
%
% This function translates a list of labels into a list of indices,
% corresponding to the locations of the labels in a master lookup table.
% Entries that aren't found have indices of NaN.
%
% "labellist" is a cell array of character vectors holding labels.
% "labellut" is a cell array of character vectors holding lookup table
%   labels. These entries should be unique (the first matching entry is
%   used in lookups).
%
% "indexlist" is a vector of the same size as "labellist" holding indices
%   into the lookup table, such that labellist(k) = labellut(indexlist(k)).
```

8.12 nlUtil_getTrimmedSampleSpan.m

```
% function [ sampfirst samplast ] = ...
%   nlUtil_getTrimmedSampleSpan( samptotalcount, trim_fraction );
%
% This picks a sample span within the range [1..samptotalcount] that trims
% the specified fraction of samples from each end.
%
% Making this a function to guarantee consistent output between instances.
%
% "samptotalcount" is the length of the span of samples to take a subset from.
% "trim_fraction" (in the range 0 to 0.5) is the fraction of the total span
%   to trim from the start and end of the span.
%
% "sampfirst" is the index of the first sample kept.
% "samplast" is the index of the last sample kept.
```

8.13 nlUtil_makePrettyTime.m

```
% function durstring = nlUtil_makePrettyTime(dursecs)
%
% This formats a duration (in seconds) in a meaningful human-readable way.
% Examples would be "5.0 ms" or "5d12h".
%
% "dursecs" is a duration in seconds to format. This may be fractional.
%
% "durstring" is a character array containing a terse human-readable
%   summary of the duration.
```

8.14 nlUtil_pruneStructureList.m

```
% function [ newlist passvec ] = ...
%   nlUtil_pruneStructureList( oldlist, filterlists )
%
% This accepts a structure array or a cell array of structures and builds a
% new list containing only those records that pass user-specified whitelist
% and blacklist filters on structure field contents.
%
% If a whitelist and a blacklist are both present for a given field, the
% whitelist is applied first, followed by the blacklist.
%
% If a field is absent, blacklist tests on it pass but whitelist tests fail.
%
% Tested field values may be scalars or character vectors.
%
```

```
% Tests can be performed on the concatenation of multiple character vector
% fields. Testing on concatenated scalar values is not supported.
%
% "oldlist" is a structure array or a cell array containing structures.
% "filterlists" is a structure array with the following fields:
%   "srcfield" is a character vector or a cell array. If it's a character
%     vector, it contains the name of the structure field to filter on. If
%     it's a cell array, it is assumed to contain one or more field names,
%     and the contents of those fields are concatenated.
%   "whitelist" is a vector or cell array containing field values to accept,
%     or [] or {} to accept all values.
%   "blacklist" is a vector or cell array containing field values to reject,
%     or [] or {} to pass all values.
%
% "newlist" is a copy of "oldlist" containing only those records that passed
%   all filters.
% "passvec" is a logical vector such that newlist = oldlist(passvec).
```

8.15 nlUtil_sparseToContinuous.m

```
% function wavedata = ...
%   nlUtil_sparseToContinuous( eventtimes, eventvalues, samprange )
%
% This converts a sequence of nonuniformly-sampled events into a continuous
% waveform. Events are assumed to reflect the first instances of changed
% waveform values; this signal is held constant after each event until the
% next event is seen.
%
% This sorts the event list by timestamp prior to processing.
%
% Event timestamps are assumed to be sample indices.
%
% "eventtimes" are the timestamps (sample indices) associated with each event.
% "eventvalues" are the data values associated with each event.
% "samprange" [min max] is the span of sample indices to generate data for.
%
% "wavedata" is a waveform spanning the specified range of sample indices,
%   where the value of any given sample is the value of the most recently-seen
%   event, or zero if there were no prior events.
```

8.16 nlUtil_sparseToContinuousTime.m

```
% function wavedata = nlUtil_sparseToContinuousTime( ...
%   eventtimes, eventvalues, firsttime, sampcount, samprate )
%
% This converts a sequence of nonuniformly-sampled events into a continuous
```

```
% waveform. Events are assumed to reflect the first instances of changed
% waveform values; this signal is held constant after each event until the
% next event is seen.
%
% This sorts the event list by timestamp prior to processing.
%
% Event timestamps are assumed to be times in seconds.
%
% "eventtimes" are the timestamps (in seconds) associated with each event.
% "eventvalues" are the data values associated with each event.
% "firsttime" is the timestamp of the first output sample to generate.
% "sampcount" is the number of output samples to generate.
% "samprate" is the sampling rate of the output waveform.
%
% "wavedata" is a waveform spanning the specified time range, where the
%   value of any given sample is the value of the most recently-seen event,
%   or zero if there were no prior events.
```

8.17 nlUtil_sprintfCellArray.m

```
% function newlist = nlUtil_sprintfCellArray( formatstr, oldlist )
%
% This applies sprintf(formatstr) to every element of a supplied cell array
% or vector, storing the resulted formatted strings in a new cell array.
%
% NOTE - Name notwithstanding, numeric input may be supplied as a vector
% as an alternative to supplying it as a cell array.
%
% "formatstr" is a character vector containing a sprintf format specifier
%   with one substitution code.
% "oldlist" is a cell array or vector containing scalars or character vectors
%   to substitute into the format specifier.
%
% "newlist" is a cell array with the same number of elements as "oldlist"
%   containing the resulting character vectors.
```

8.18 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.
```

8.19 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.
```

8.20 nlUtil_testXMLStructAttributes.m

```
% function ismatch = ...
%   nlUtil_testXMLStructAttributes( xmlstruct, attribswanted )
%
% This searches through an XML parse structure (returned by readstruct()),
% and determines whether the top-level node matches the desired attribute
% criteria.
%
% "xmlstruct" is a structure containing an XML parse tree (per readstruct()).
% "attribswanted" is a cell array containing names and values of attributes
%   to match. If _all_ specified attributes match _any_ of their permitted
%   values, or if "attribswanted" is empty, the structure matches. The cell
%   array has the form { 'attr1', { 'val1', 'val2' }, 'attr2', ... }. Matched
%   values are character arrays and are not sensitive to case.
%
% "ismatch" is true if "xmlstruct" matches the desired attribute criteria
%   and false otherwise.
```

Part II

Abstraction Libraries

Chapter 9

Compatibility Library Structures and Additional Notes

9.1 FOLDERMETA_INTAN.txt

```
Intan-specific folder metadata format is as follows. This is intended as a
reference for maintaining code; nothing outside of this set of functions
should need to look at Intan-specific metadata.

The sole exception is knowing what to look for in the "devicetype" field.



In "folder metadata":

- "devicetype" is 'intan'.
- "nativemeta" is the output of nlIntan_readMetadata().
- Banks that may exist are as follows:
  - "AmpA".."AmpH" contain ephys channel signals.
  - "AuxA".."AuxH" contain on-chip aux analog input signals.
  - "DcA".."DcH" contain low-gain DC-coupled ephys channel signals.
  - "Ain" and "Aout" contain BNC analog inputs and outputs, respectiely.
  - "Din" and "Dout" contain TTL inputs and outputs, respectively.
  - "StimA".."StimH" contain stimulation drive currents.
  - "FlagsA".."FlagsH" contain encoded stimulation-related flags.
  - "VddA".."VddH" contain voltage supply signals.



In "signal bank metadata":

- "banktype" is:
  - 'boolean' for "Din" and "Dout", if they're stored per-channel.
  - 'integer' for "Din" and "Dout" otherwise.
```

```
  - 'flagvector' for "FlagsA".."FlagsH".
  - 'analog' for other banks.
- Ephys recording channels have "fpunits" of 'uV'; stimulation channels have
  "fpunits" of 'uA'. Other analog channels have "fpunits" of 'V'. Channels
  that aren't analog have "fpunits" of ''.
- "handle" is a structure with the following fields:

  - "format" is 'onefileperchan', 'neuroscope', or 'monolithic'.
  - "special" is '', 'stimflags' or 'stimcurrent'.

  For "monolithic":
    FIXME - Monolithic NYI.

  For "neuroscope":
    FIXME - Neuroscope NYI.

  For "onefileperchan":
  - "chanfilechans" is a vector containing channel indices in the same order
    as "chanfilenames".
  - "chanfilenames" is a vector containing channel data filenames in the
    same order as "chanfilechans".
  - "timefile" is the file to read sample indices from.



This is the end of the file.
```

9.2 FOLDERMETA_OPENEPHYS.txt

```
Open Ephys-specific folder metadata format is as follows. This is intended
as a reference for maintaining code; nothing outside of this set of functions
should need to look at Open Ephys-specific metadata.

The sole exception is knowing what to look for in the "devicetype" field.


FIXME - This may need to be updated if support for one file per channel
format and for spike data are added.



In "folder metadata":

- "devicetype" is 'openephys'.
- "firsttime" is the smallest timestamp value seen across all banks, in
  native format. The I/O functions subtract this from all per-bank timestamps.
```

```
In "signal bank metadata":

- "nativemeta" is the "header" field from the output of
    "load_open_ephys_binary()".
- "banktype" is:
  - 'analog' for continuous data.
  - 'eventwords' for the read-as-words alias of event data.
  - 'eventbool' for the read-as-bits alias of event data.
- "firsttime" is the smallest timestamp value seen in this bank, in native
  format.



This is the end of the file.
```

9.3 FT_EVENTS.txt

```
Field Trip events are constructed using channels that had type "eventbool"
or "eventwords" (i.e. channels that are stored as sparse rather than
continuous time series).

Field Trip's event record fields that we set are "sample" (event location in
samples), "value" (event value in the native channel type), and "type" (set
to the channel type: "eventbool" or "eventwords"). Other fields are set to
the empty value ([]).

Field Trip does not appear to support event filtering based on source (the
event's channel). To do this, use the "nlFT_selectChannels()" function before
calling "ft_read_events()". There should be exactly one channel selected;
results from selecting multiple channels are undefined.


(This is the end of the file.)
```

9.4 FT_ITERFUNC.txt

```
An iteration processing function handle is called to perform signal
processing when iterating across trials and channels within a Field Trip
data structure. The intention is to simplify processing of Field Trip data
using non-FT functions.


An iteration processing function has the form:

[ waveresult otherresult ] = ...
```

```
  iterfunc( wavedata, timedata, samprate, trialidx, chanidx, chanlabel )

"wavedata" is a vector containing the waveform to be processed (from the
  Field Trip data's "trial" field).
"timedata" is a vector containing sample times (from the Field Trip data's
  "time" field).
"samprate" is the sampling rate (from the Field Trip data's "fsample" field).
"trialidx" is the trial number.
"chanidx" is the channel number.
"chanlabel" is the corresponding channel label (from the Field Trip data's
  "label" field).

"waveresult" is a vector containing modified waveform data. This is
  typically used to build a modified version of the "trial" cell array.
"otherresult" is an arbitrary data type containing any other information
  that the user wishes to associate with this input data/trial/channel.


A typical iteration processing functiton definition would be as follows.
This example wraps a helper function that is passed additional arguments
set at the time the processing function is defined.

tuning_parameters = (stuff);
other_parameters = (stuff);
iterfunc = @( wavedata, timedata, samprate, trialidx, chanidx, chanlabel ) ...
  helper_do_iteration_processing( wavedata, timedata, samprate, ...
    trialidx, chanidx, chanlabel, tuning_parameters, other_parameters );


This is the end of the file.
```

9.5 FT_ITERFUNC_BATCHED.txt

```
A batched iteration processing function handle is called to perform signal
processing when iterating across a very large FT dataset in batches of
channels or batches of trials (to avoid loading the entire dataset into
memory).

The intention is to abstract away the channel and trial batching logic.

A batched iteration processing function has the form:

[ ftdata_new auxdata ] = ...
  iterfunc_batched( ftdata_old, chanindices_orig, trialindices_orig )

"ftdata_old" is a "ft_datatype_raw" structure containing this batch's data.
"chanindices_orig" is a vector with one entry per channel in ftdata_old
  containing each channel's corresponding index in the larger dataset
```

```
  (prior to batching).
"trialindices_orig" is a vector with one entry per trial in ftdata_old
  containing each trial's corresponding index in the larger dataset
  (prior to batching).

"ftdata_new" is a modified version of ftdata_old (after any iterator
  data processing is performed), or struct([]) to omit consturction.
"auxdata" is a cell array indexed by {trial, channel} containing arbitrary
  user-defined data that the user wishes to associate with each trial/channel.


A typical batched iteration processing function definition would be as
follows. This example wraps a helper function that is passed additional
arguments set at the time the processing function is defined, and also
wraps nlFT_iterateAcrossData() for per-trial/per-channel processing.


tuning_parameters = (stuff);
other_parameters = (stuff);
processing_config = (stuff);

iterfunc_batched = @( ftdata_old, chanindices_orig, trialindices_orig ) = ...
  helper_do_batch_iteration( ftdata_old, ...
    chanindices_orig, trialindices_orig ...
    processing_config, tuning_parameters, other_parameters );


function [ ftdata_new auxdata ] = helper_do_batch_iteration( ...
  ftdata_old, chanindices_orig, trialindices_orig, ...
  proc_config, tuning_params, other_params )

  ftdata_new = ft_preprocessing(proc_config, ftdata_old);

  iterfunc_single = @( wavedata, timedata, samprate, ...
    trialidx, chanidx, chanlabel ) ...
    helper_do_channel_iteration( wavedata, timedata, samprate, ...
      trialidx, chanidx, chanlabel, chanindices_orig, trialindices_orig, ...
      tuning_params, other_params );

  [ newtrials auxdata ] = ...
    nlFT_iterateAcrossData( ftdata_new, iterfunc_single );
  ftdata_new.trial = newtrials;

end


This is the end of the file.
```

9.6 INTAN_FILES.txt

```
Data is saved in monolithic format ("traditional Intan"), one-file-per-type
format (NeuroScope-compatible), or one-file-per-channel format.



"Traditional Intan" format stores data as follows:

- Data is stored as a self-contained ".rhd" file (for the recording
  controller) or ".rhs" file (for the stimulate-and-record controller).

- A session may be saved as multiple independent files (with a new file
  started every N minutes).

- All ".rhd" and ".rhs" files begin with a header containing metadata.

- Blocks of data follow the header.

- For file formats other than "traditional Intan", an "info.rhd" or
  "info.rhs" file exists with a header and no blocks of data.

See Intan's file format documentation for stucture details for the header
and the data blocks.



"One-file-per-type" format (NeuroScope-compatible) stores signals in the
following files:

- All files are saved at the full sampling rate for ease of alignment, even
  if the underlying signals are sampled at lower rates (as with auxiliary
  and power supply signals).

- "time.dat" is a series of (signed int32) sample indices.

- "amplifier.dat" stores ephys channel signals (signed int16, 0.195 uV/LSB).

- "auxiliary.dat" (RHD only) stores on-chip aux analog input signals
  (unsigned uint16, 37.4 uV/LSB)

- "dcamplifier.dat" (RHS only) stores low-gain DC-coupled ephys channel
  signals (unsigned uint16, 19.23 mV/LSB, zero level 512).

- "analogin.dat" stores BNC analog input signals (unsigned uint16,
  0.3125 mV/LSB for RHS/RHD, 50.354 uV/LSB mode 0, 0.15259 mV/LSB mode 1,
  zero level 32768).

- "analogout.dat" (RHS only) stores BNC analog output signals (unsigned
```

```
  uint16, 0.3125 mV/LSB, zero level 32768).

- "digitalin.dat" stores TTL input signals as a packed unsigned 16-bit word.

- "digitalout.dat" stores TTL output signals as a packed unsigned 16-bit word.

- "stim.dat" (RHS only) stores stimulation current and flags as encoded
  unsigned 16-bit values. The least-significant 8 bits are the current
  magnitude (units per the header), the 9th bit is the sign, the 14th bit is
  amplifier settle, the 15th bit is the charge recovery, and the 16th bit is
  compliance limit. Note that bit numbering starts at 1, not 0.

- "supply.dat" (RHD only) stores on-chip supply voltage sensor signals
  (unsigned uint16, 74.8 uV/LSB)



"One-file-per-type" format (NeuroScope-compatible) stores signals in the
following files:

- File format is per "one-file-per-type", except with only one channel per
  file, unless otherwise indicated.

- All files are saved at the full sampling rate for ease of alignment, even
  if the underlying signals are sampled at lower rates (as with auxiliary
  and power supply signals).

- Note that as of 2021 the stim/record controller (RHS) only has four banks
  ("A".."D") and a maximum of 32 channels per bank ("000".."031").

- "time.dat" is a series of (signed int32) sample indices.

- "amp-X-NNN.dat" files store ephys channel signals for bank X ("A".."H")
  and channel NNN ("000".."127").

- "aux-X-AUXN.dat" files (RHD only) store on-chip aux analog input signals
  for bank X ("A".."H") and channel N ("1".."6"). Channels 1..3 are for the
  first connected chip and channels 4..6 are for the second connected chip
  on that bank.

- "dc-X-NNN.dat" files (RHS only) store low-gain DC-coupled ephys channel
  signals for bank X ("A".."H") and channel NNN ("000".."127").

- "board-ADC-NN.dat" files store BNC analog input signals for
  channel NN ("00".."07").
- "board-ANALOG-IN-NN.dat" is an alternate set of filenames for this.
- FIXME - Saved as 1..8, not 0..7!

- "board-ANALOG-OUT-N" files store BNC analog output signals for
  channel N ("0".."7").
```

```
- FIXME - Saved as 1..8, not 0..7!

- "board-DIN-NN.dat" files store TTL input signals for channel NN
  ("00".."15"). Samples are still unsigned uint16 values but only have
  values of 0 or 1.
- "board-DIGITAL-IN-NN.dat" is an alternate set of filenames for this.
- FIXME - Saved as 1..16, not 0..15!

- "board-DOUT-NN.dat" files store TTL output signals for channel NN
  ("00".."15"). Samples are still unsigned uint16 values but only have
  values of 0 or 1.
- "board-DIGITAL-OUT-NN.dat" is an alternate set of filenames for this.
- FIXME - Saved as 1..16, not 0..15!

- "stim-X-NNN.dat" files store encoded stimulation current and flags for
  bank X ("A".."H") and channel NNN ("000".."127").

- "vdd-X-VDDN.dat" files (RHD only) store on-chip supply voltage sensor
  signals for bank X ("A".."H"). "VDD1" is for the first connected chip and
  "VDD2" is for the second connected chip on that bank.



This is the end of the file.
```

9.7 INTAN_METADATA.txt

```
An Intan metadata structure is a structure containing header information
supplied by RHD and RHS files, in a format consistent with that provided by
"read_Intan_RHD2000_file.m" and "read_Intan_RHS2000_file.m".

The metadata structure has the following fields:

"filename" is the file that was read (including path).
"path" is the folder path containing data files.
"devtype" is 'RHD' or 'RHS'.
"version_major" is the major version number of the data file.
"version_minor" is the minor version number of the data file.
"num_samples_per_data_block" is the number of samples per data block, for
  monolithic data.
"frequency_parameters" is a structure with the following fields:
  "amplifier_sample_rate" is the ephys channel sampling rate.
  "aux_input_sample_rate" is the sampling rate of the on-chip auxiliary
    analog inputs.
  "supply_voltage_sample_rate" is the sampling rate of the chip supply
    voltage measurement.
  "board_adc_sample_rate" is the sampling rate of the RHD controller's
    BNC analog inputs.
```

```
  "board_dig_in_sample_rate" is the sampling rate of the RHD controller's
    TTL inputs.
  "desired_dsp_cutoff_frequency" is the user-specified cutoff for the
    on-chip first-order digital high-pass filter.
  "actual_dsp_cutoff_frequency" is the implemented cutoff for the
    on-chip first-order digital high-pass filter.
  "dsp_enabled" is nonzero if the on-chip digital filter is active.
  "desired_lower_bandwidth" is the user-specified lower bandwidth for
    the ephys amplifiers. This is a first-order high-pass filter.
  "actual_lower_bandwidth" is the implemented lower bandwidth for the
    ephys amplifiers. This is a first-order high-pass filter.
  "desired_lower_settle_bandwidth" (RHS-only) is the user-specified lower
    bandwidth for the ephys amplifiers when recovering from stimulation.
  "actual_lower_settle_bandwidth" (RHS-only) is the implemented lower
    bandwidth for the ephys amplifiers when recovering from stimulation.
  "desired_upper_bandwidth" is the user-specified upper bandwidth for
    the ephys amplifiers. This is a third-order low-pass filter.
  "actual_upper_bandwidth" is the implemented upper bandwidth for the
    ephys amplifiers. This is a third-order low-pass filter.
  "notch_filter_frequency" is the center frequency of the power line
    frequency rejection filter, or 0 for no notch filtering. This is
    a software-implemented biquad filter.
  "desired_impedance_test_frequency" is the user-specified frequency
    used for measuring channel impedance.
  "actual_impedance_test_frequency" is the implemented frequency used
    for measuring channel impedance.
"stim_parameters" (RHS only) is a structure with the following fields:
  "stim_step_size" is the current scale used for stimulation (amps/LSB).
  "charge_recovery_current_limit" is the current driven when restoring the
    specified voltage during charge recovery (in amperes).
  "charge_recovery_target_voltage" is the voltage to settle stimulation
    channels to after stimulation, if performing charge recovery (in volts).
  "amp_settle_mode" indicates which method was used for fast-settling
    after stimulation. 0 uses bandwidth-switching (recommended) and 1 resets
    the on-chip amplifiers.
  "charge_recovery_mode" indicates which method of charge recovery was used.
    0 uses current-limited charge recovery and 1 uses a resistive switch.
"notes" is a structure with the following fields:
  "note1" is a character array containing the first notes string.
  "note2" is a character array containing the second notes string.
  "note3" is a character array containing the third notes string.
"num_temp_sensor_channels" is the number of temperature sensor channels.
"dc_amp_data_saved" is nonzero if low-gain DC-coupled amplifier signals were
  saved for each channel (suitable for monitoring stimulation artifacts).
"board_mode" indicates board configuration. This mostly affects BNC
  analog input scale. Mode 0 is 50.354 uV/LSB, mode 1 is 152.59 uV/LSB,
  and mode 13 is 312.5 uV/LSB. These correspond to ranges of 0..3.3V,
  +/-5V, and +/-10.24V.
"voltage_parameters" is a structure with the following fields:
  "amplifier_scale" is the ephys voltage scale in V/LSB.
```

```
  "aux_scale" is the chip auxiliary input scale in V/LSB.
  "dcamp_scale" is the RHS low-gain amplifier scale in V/LSB.
  "dcamp_zerolevel" is the RHS low-gain amplifier output with 0V input.
  "board_analog_scale" is the BNC analog input and output scale in V/LSB.
  "board_analog_zerolevel" is the BNC analog I/O value with a 0V signal level.
  "supply_scale" is the RHD supply voltage scale in V/LSB.
  "temperature_scale" is the RHD temperature sensor scale in degC/LSB.
"reference_channel" is a character array containing the name of the
  channel used for re-referencing the input, or an empty string if no
  re-referencing was performed.
"amplifier_channels" is a structure array with zero or more entries
  containing ephys channel metadata.
"spike_triggers" is a structure array with the same number of entries
  as "amplifier_channels", containing spike scope trigger settings for
  each ephys channel.
"aux_input_channels" is a structure array with zero or more entries
  containing ephys chip auxiliary analog input channel metadata.
"supply_voltage_channels" is a structure array with zero or more entries
  containing ephys chip supply voltage monitor channel metadata.
"board_adc_channels" is a structure array with zero or more entries
  containing controller BNC analog input channel metadata.
"board_dig_in_channels" is a structure array with zero or more entries
  containing controller TTL input channel metadata.
"board_dig_out_channels" is a structure array with zero or more entries
  containing controller TTL output channel metadata.
"num_data_blocks" is the number of blocks of monolithic data stored. This is
  zero for NeuroScope or per-channel data.
"header_bytes" is the number of bytes to skip before reading the first
  block of monolithic data.
"bytes_per_block" is the number of bytes per monolithic data block.


Channel metadata structures contain the following fields:

"native_channel_name" is the device-specific channel name (e.g. 'A-015').
"custom_channel_name" is a user-defined channel name (e.g. 'Probe 15').
"native_order" is the device-assigned channel number.
"custom_order" is a user-assigned channel number.
"board_stream" is the device-assigned internal data stream number.
"chip_channel" is the on-chip channel number.
"port_name" is a human-readable name for this channel's port (e.g. 'Port A').
"port_prefix" is a device-assigned label for this channel's port (e.g. 'A').
"port_number" is a device-assigned port index (ports are numbered 1..N).
"electrode_impedance_magnitude", if applicable, is the measured
  impedance magnitude of this channel (in ohms).
"electrode_impedance_phase", if applicable, is the measured impedance
  phase angle of this channel (in degrees).


Spike scope trigger setting structures contain the following fields:
```

"voltage_trigger_mode" is 0 for TTL triggered and 1 for threshold-triggered.
"voltage_threshold" is the trigger threshold in uV.
"digital_trigger_channel" is the TTL trigger input number (0-15).
"digital_edge_polarity" is 0 for falling-edge triggered and 1 for rising-edge.

This is the end of the file.

9.8 OPENEPHYS_CHANMAP.txt

This file documents Matlab structures used to represent configurations of
the ChannelMappingNode plugin in Open Ephys.

In Open Ephys v0.5.x, this also includes referencing information. In Open
Ephys v0.6.x, re-referencing is moved to its own plugin.

An Open Ephys v5 channel map is represented as a structure with the following
fields:

"oldchan" is a vector indexed by new channel number containing the old
  channel number that maps to each new location, or NaN if none does.

"oldref" is a vector indexed by new channel number containing the old
  channel number to be used as a reference for each new location, or NaN
  if unspecified.

"isenabled" is a vector of boolean values indexed by new channel number
  indicating which new channels are enabled.

(This is the end of the file.)

9.9 OPENEPHYS_DATA.txt

Miscellaneous notes about how Open Ephys stores things (in the Womelsdorf Lab
setup) and about Open Ephys's I/O functions.

Open Ephys's I/O functions: https://github.com/open-ephys/analysis-tools/

Open Ephys has two formats:
- "Open Ephys" format, with one file per channel. This is deprecated.

- "Binary" format, with one file for all continuous data.

"Open Ephys" format is structured as follows:

- Each session just produces one directory.
- There's a "settings.xml" file in the session directory.
- There's a "Continuous_Data.openephys" file with metadata.
- There are "(label).continuous" files with the continuous data itself.
- There are a small number of ".events" files with monolithic event data.

"Binary" format is structured as follows:

- Each session gets a deep directory tree.
- There's a "settings.xml" file in a session's root directory.
- For each experiment and recording node, there's a "structure.oebin" file
  that describes what that node recorded.
- There are directory trees storing continuous data, event data, and spike
  data from devices attached to the recording node. Any given device
  produces a small number of monolithic data files (data, timestamps,
  event states, etc).

Open Ephys has several I/O functions, with a few peculiarities:

- "list_open_ephys_binary" is pointed at a directory containing a ".oebin"
  file. It returns a cell array containing the names of subfolders which have
  the type of data requested. Zero, one, or multiple folders may exist.
- "load_open_ephys_binary" loads folder number N of a requested type from
  the list produced by "list_open_ephys_binary". This returns header
  information with metadata, and either data or a memory-mapped file handle.
  You want to memory-map this data; it's usually too big for RAM.

- "get_session_info" is pointed at a directory containing "settings.xml".
  This is supposed to read configuration metadata, but in practice it only
  does this for sessions stored in the older "Open Ephys" format. So, you're
  flying blind with ".oebin" binary format.
- "load_open_ephys_data" and its variants are intended to read data stored
  in the older "Open Ephys" format. I haven't tested these, since we only use
  the new format.

Event data (from "binary" format) is structured as follows:

- Event lists are stored as .npy tables in subfolders indicating the event
  type which are in turn in subfolders indicating the devices of origin.
- TTL events have the following fields:
  - Timestamps (in samples).
  - ChannelIndex (channel number that changed)

```
  - Data (+chan for rising edges or -chan for falling)
  - FullWords (Nx2 uint8 matrix with word bytes; 1 = least significant)
- This data is redundant; ChannelIndex == abs(Data), and FullWords can be
  reconstructed from data and vice versa.
- Multiple bits changing at the same time results in multiple events with
  the same timestamp.

- FIXME - No information about what's in text events, as these read as empty.


(This is the end of the file.)
```

9.10 OPENEPHYS_TTLWORDS.txt

```
Something very important to keep in mind about Open Ephys's TTL events:

Individual TTL bit lines are always correct and consistent, but the word data
associated with these events is _not_ always consistent.


The problem happens because Open Ephys generates an event every time a bit
changes, so if multiple bits change during one sample, multiple events with
the same timestamp are generated. The "FullWords" vector is updated after
each event, one bit at a time - so events with the same timestamp will have
"FullWords" vectors with different content. The events are not necessarily
returned in the same order that they were processed in, so there's no easy
way to tell which "FullWords" value is the right one in a set of events
that have the same timestamp.


The right thing to do is to either build your own word data and ignore
the words provided by OpenEphys, or else to use additional knowledge about
the experiment scenario to disambiguate these cases.


(This is the end of the file.)
```

9.11 PROCMETA_OPENEPHYSv5.txt

```
Processor node metadata from Open Ephys v5 is saved as structures (one per
processor node), derived from the "processor" tag parse trees returned by
readstruct(). These metadata structures have the following fields:


Common to all processor node types:
```

"rawconfig" is the raw XML parse tree for the processor node, as returned
  by "readstruct()".

"procname" is a character vector containing the "pluginName" attribute.
"proclib" is a character vector containing the "libraryName" attribute.
"procnode" is a number containing the "NodeId" attribute.

"channelselect" is a logical vector containing the channel selection state's
  "param" attribute values from all "channel" tags in the processor node.
  This may be an empty vector for special nodes like splitters.
  Note that the "number" attribute starts counting at 0, so the index in
  "channelselect" is equal to "number" + 1.
"eventbanks" is an integer indicating the number of <EVENTCHANNEL> tags
  found in this processor node.

"descsummary" is a cell array of character vectors containing a
  human-readable short summary of the node configuration. This may be {}.
"descdetailed" is a cell array of character vectors containing a
  human-readable detailed description of the node configuration. This
  may be {}.

For nodes with plugin name 'Intan Rec. Controller':

"samprate" is the sampling rate used by the recording controller, in Hz.
"bandpass" [low high] describes the corner frequencies of the headstage
  band-pass filters connected to the controller.
"chanlabels" is a cell array containing the names of the channels
  exported by the recording controller (in Open Ephys's order).

For nodes with plugin name 'Channel Map':

"chanmap" is a structure containing a channel map, per OPENEPHYS_CHANMAP.txt.

For nodes with plugin name 'Record Node':

"writefolder" is a character vector storing the folder path to save to.
"wantevents" is true if events are to be saved, false otherwise.
"wantspikes" is true if spikes are to be saved, false otherwise.
"savedchans" is a logical vector indicating whether each channel is to be
  saved (FIXME: out of those accepted by channelselect?).

```
For nodes with plugin name 'File Reader':


"filename" is a character vector storing the file being read from.
"sampstart" is the first sample number in the file.
"sampstop" is the last sample number in the file.
"chancount" is the number of channels saved.

NOTE - Sampling rate and channel names aren't saved in the config state.
All of that has to be found by opening and parsing the file.



For nodes with plugin name 'Arduino Output':


"serialdev" is a character vector with the name of the serial device the
  Arduino is connected to.
"ardoutput" is the Arduino digital I/O number used for output.

"inputpretty" is the native name of the TTL input being monitored. It has
  the form "channel:bit (channel name)".
"inputbank" is the channel (bank) number of the TTL input being monitored,
  starting at 0.
"inputbit" is the sub-channel (bit) number of the TTL input being monitored,
  starting at 0.
"inputlabel" is a character vector with the channel (bank) name of the TTL
  input being monitored.

"gatepretty" is the native name of the TTL input being used as a gate. It has
  the form "channel:bit (channel name)".
"gatebank" is the channel (bank) number of the TTL input being used as a gate,
  starting at 0.
"gatebit" is the sub-channel (bit) number of the TTL input being used as a
  gate, starting at 0.
"gatelabel" is a character vector with the channel (bank) name of the TTL
  input being used as a gate.



For nodes with plugin name 'Bandpass Filter':


"band" [ low high ] describes the filter corner frequencies in Hz.
```

For nodes with plugin name 'Phase Calculator' (TNE Lab):

"bandcorners" [low high] describes the corner frequencies of the phase
  calculator's bandpass filter.
"bandedges" [low high] describes the nominal cutoff frequencies of the
  phase calculator's bandpass fitler.

"predictorder" is the order of the AR predictor used by the phase calculator.
"predictupdatems" is the update interval (in ms) for training the predictor.

"outputmode" is a character vector indicating which outputs are produced
  by the phase calculator. Valid values are 'phase', 'magnitude', and 'both'.
  Single outputs replace the input channel; 'both' adds an extra channel
  for magnitude and replaces the input channel with phase.

For nodes with plugin name 'Crossing Detector' (TNE Lab):

"inputchan" is the ephys channel number to monitor.
"wantrising" is true if triggers happen when crossing low-to-high.
"wantfalling" is true if triggers happen when crossing high-to-low.

"outputTTLchan" is the (1-based) number of the output TTL bit to generate
  events on. This is in a new TTL bank (new "event channel").

"threshtype" is a character vector indicating the method by which thresholding
  is performed. Valid values are 'constant', 'random', 'channel', 'adaptive',
  and 'averagemult'.

"threshold" is the threshold value used for 'constant' thresholding, and the
  multiplier used for 'averagemult' thresholding.
"randomrange" [min max] is the range threshold values are drawn from when
  using 'random' thresholding (uniform random sampling within that range span).
"extthreshchan" is the analog channel number to use as an externally supplied
  threshold with 'channel' triggering.
"averageseconds" is the approximate smoothing time used for averaging with
  'averagemult' thresholding.

Adaptive thresholding receives an input "indicator" signal and adjusts its
threshold until the input signal meets a target value. Its configuration
parameters are:

"adaptinputname" is a character vector containing the label of the input
  channel to monitor as an "indicator".
"adapttarget" is the value that the algorithm tries to get the indicator
  input to meet.
"adaptinputrange" [min max] is the range to which input values should be

```
  clamped, or [] to not perform clamping.
"adaptoutputrange" [ min max ] is the range to which adjusted threshold
  values should be clamped, or [] to not perform clamping.
"adaptlearnratestart" is the initial learning rate to use when adjusting.
"adaptlearnratemin" is the value to which the learning rate decays towards.
"adaptlearnratedecay" is the decay rate for the learning rate. See the TNE
  Lab web page for a detailed discussion of the algorithm.



For nodes with plugin name 'TTL Cond Trigger' (ACC Lab):


"outcount" is the number of outputs (always 4, for now).
"incount" is the number of inputs per output (always 4, for now).

"outconditions" is a cell array with one structure per output; each structure
  has the following fields:

  "enabled" is true if the output is being used and false otherwise.
  "anyall" is 'any' or 'all', indicating how inputs are combined.
  "name" is a character vector containing the output name.

  (Logic processing fields are also present, described below.)

  "inconditions" is a cell array with one structure per input; each structure
    has the following fields:

    "enabled" is true if the input is being used and false otherwise.
    "digbankidx" is the input TTL bank (channel) number, starting at 0.
    "digbit" is the input TTL bit (sub-channel) number, starting at 0.
    "name" is a character vector containing the input name.

    (Logic processing fields are also present, described below.)


Logic processing fields are a set of fields that define how a signal's
output is derived from its input:

  "trigtype" is 'high' (level-triggered active-high), 'low' (level-triggered
    active-low), 'rising' (edge-triggered), or 'falling' (edge-triggered).
  "delayminsamps" and "delaymaxsamps" define the time to wait from the
    trigger conditions being met to the output signal being asserted. For any
    given trigger event, this is uniformly sampled from the specified range.
  "sustainsamps" is the number of samples for which the output is to be
    asserted.
  "deadsamps" is the number of samples that have to elapse between one
    detected trigger event and the next detected trigger event. Detections
    that happen during the dead time are ignored.
  "deglitchsamps" is the number of samples for which the input must be stable
```

```
    before a trigger event is recognized. It must be less than or equal to
    "delayminsamps".
  "activehigh" is true if the output is driven high when asserted, and false
    if the output is driven low when asserted.



Unrecognized plugin names will still return the "common to all types"
metadata, which includes the full parse tree, facilitating user processing
of unrecognized configuration information.



(This is the end of the file.)
```

Chapter 10

"nlFT" Functions

10.1 nlFT_applyNaNMask.m

```
% function newftdata = nlFT_applyNaNMask( oldftdata, trialmask )
%
% This function NaNs out samples in Field Trip trials following the pattern
% recorded in "trialmask".
%
% This is intended to be used with "nlFT_getNaNMask" and "nlFT_fillNaN".
% NaN segments are interpolated, filtering is performed, and then NaN
% segments are restored.
%
% "oldftdata" is a ft_datatype_raw dataset to modify.
% "trialmask" is a cell array with NaN masks, per nlFT_getNaNMask().
%
% "newftdata" is a copy of "oldftdata" with indicated samples set to NaN.
```

10.2 nlFT_applyTimeWindowSquash.m

```
% function newftdata = nlFT_applyTimeWindowSquash( oldftdata, timemasks )
%
% This NaNs out the indicated time regions in every trial within a Field
% Trip dataset.
%
% This is intended to be used with nlFT_getWindowsAroundEvents for artifact
% rejection.
%
% "oldftdata" is a ft_datatype_raw dataset to modify.
% "timemasks" is a 1xNtrials cell array. Each cell contains a 1xNsamples
%   logical vector that's true on every sample to be squashed.
%
% "newftdata" is a copy of "oldftdata" with the indicated samples set to NaN.
```

10.3 nlFT_calcSteppedWindowSpectrogram.m

```
% function [ freqlist timelist spectpowers ] = ...
%   nlFT_calcSteppedWindowSpectrogram( ...
%     ftdata, winsize, winstep, timespan, freqspan )
%
% This computes a spectrogram of Field Trip data using a stepped rectangular
% window.
%
% Trials are assumed to all include the requested time span.
%
% "ftdata" is a ft_datatype_raw structure containing signal data.
% "winsize" is the window duration in seconds.
% "winstep" is the window step distance in seconds.
% "timespan" [ min max ] is the time region within which the window is to be
%   stepped.
% "freqspan" [ min max ] is the range of frequencies to evaluate.
%
% "freqlist" is a vector containing frequencies for which power was computed.
% "timelist" is a vector containing window center times that were evaluated.
% "spectpowers" is a nTrials x nChannels x nFrequencies x nTimes matrix
%   containing evaluated spectral power.
```

10.4 nlFT_compareChannelMaps.m

```
% function [ firstok secondok logtext ] = nlFT_compareChannelMaps( ...
%   firstsrc, firstdst, secondsrc, seconddst )
%
% This compares two label-based channel maps and flags discrepancies.
%
% "firstsrc" is a cell array of "before mapping" labels for the first map.
% "firstdst" is a cell array of "after mapping" labels for the first map.
% "secondsrc" is a cell array of "before mapping" labels for the second map.
% "seconddst" is a cell array of "after mapping" labels for the second map.
%
% "firstok" is a vector of the same size as "firstsrc" containing "true" for
%   entries that are consistent between the first and second maps.
% "secondok" is a vector of the same size as "secondsrc" containing "true"
%   for entries that are consistent between the first and second maps.
% "logtext" is a character vector containing a human-readable summary report.
```

10.5 nlFT_compressFTEvents.m

```
% function [ evtable newlut ] = nlFT_compressFTEvents(evstructarray, typelut)
%
```

```
% This compresses a Field Trip event list by converting type labels into
% numbers, removing "offset" and "duration" if empty, and storing the result
% as a table. MatLab structures have more overhead and MatLab cell arrays
% have a _lot_ of overhead, so the result takes up much less memory.
%
% This is intended to be used with event lists generated by the LoopUtil
% library, which store TTL state changes as events with "type" holding the
% channel label.
%
% FIXME - Fallback for numeric rather than character "type" data is to copy
% the type field without translation and return an empty LUT.
%
% "evstructarray" is the Field Trip event list (structure array).
% "typelut" is a cell array containing values expected in the "type" field.
%   For event lists generated by the LoopUtil library, this should be the
%   "label" field from the Field Trip header (channel label list).
%
% "evtable" is a table containing data from the event list.
% "newlut" is a copy of "typelut" with additional entries for any "type"
%   values that weren't recognized.
```

10.6 nlFT_evalChannelMapFit.m

```
% function logtext = nlFT_evalChannelMapFit( ...
%   srclabels, dstlabels, data_before, data_after, rmatrix, pmatrix )
%
% This evaluates how well a supplied channel mapping matches correlations
% between channels in "before" and "after" datasets.
%
% The datasets must have the same sampling rate, the same number of trials,
% and the same number of samples in corresponding trials.
%
% NOTE - Computing correlation values is slow! Time goes up as the square of
% the number of channels.
%
% "srclabels" is a cell array containing channel labels from "data_before".
% "dstlabels" is a cell array containing corresponding channel labels from
%   "data_after".
% "data_before" is a Field Trip dataset prior to channel mapping.
% "data_after" is a Field Trip dataset after channel mapping.
% "rmatrix" is the matrix of R-values returned by nlFT_getChannelCorrelMatrix.
%   If this argument is omitted, the R-value matrix is recomputed.
% "pmatrix" is the matrix of P-values returned by nlFT_getChannelCorrelMatrix.
%   If this argument is omitted, the P-value matrix is recomputed.
%
% "logtext" is a character vector containing a human-readable summary report.
```

10.7 nlFT_eventListToWaveform.m

```
% function wavedata = ...
%   nlFT_eventListToWaveform( ftevents, desiredlabel, samprange )
%
% This processes a Field Trip event list and converts it to a waveform.
%
% "ftevents" is a vector of Field Trip event records. Relevant fields are
%   "value", "sample", and "type" (containing the channel label).
% "desiredlabel" is a character vector containing the value to match to the
%   events' "type" field. If this is empty, all events are accepted.
% "samprange" [ min max ] is the span of sample indices to generate data for.
%
% "wavedata" is a waveform spanning the specified range of sample indices,
%   where the value of any given sample is the value of the most
%   recently-seen event, or zero if there were no prior events.
```

10.8 nlFT_extendNaNQuartile.m

```
% function newftdata = nlFT_extendNaNQuartile( oldftdata, threshold )
%
% This segments signals into NaN and non-NaN regions, and extends the NaN
% regions to cover adjacent samples that are sufficiently large excursions
% from their host non-NaN regions.
%
% This is intended to extend artifact-squash regions to cover portions of
% the artifact that are still present after squashing.
%
% This is a wrapper for nlArt_extendNaNQuartile.
%
% "oldftdata" is a ft_datatype_raw dataset to process.
% "threshold" is the amount by which the signal must depart from the median
%   level to be considered a residual artifact. This is a multiple of the
%   median-to-quartile distance (about two thirds of a standard deviation).
%   Typical values are 6-12 for clear exponential excursions. If this is
%   NaN or Inf, no squashing is performed.
%
% "newftdata" is a copy of "oldftdata" with NaN regions extended to cover
%   adjacent excursions.
```

10.9 nlFT_fillNaN.m

```
% function newftdata = nlFT_fillNaN( oldftdata )
%
% This calls nlProc_fillNaN() to interpolate NaN segments within all trials
```

```
% of the supplied Field Trip dataset.
%
% "oldftdata" is a ft_datatype_raw dataset to modify.
%
% "newftdata" is a copy of "oldftdata" with NaN segments interpolated.
```

10.10 nlFT_findChannelIndices.m

```
% function newindices = nlFT_findChannelIndices( ftheader, chanlabels )
%
% This returns a vector of Field Trip channel indices corresponding to the
% specified list of Field Trip channel labels.
%
% NOTE - If channel labels aren't unique, a matching channel label is chosen
% arbitrarily. If channel labels aren't found in the FT header, an index of
% NaN is returned for that channel.
```

10.11 nlFT_findSpectrumPeaks.m

```
% function [ peakfreqs peakheights peakwidths ] = ...
%   nlFT_findSpectrumPeaks( ftdata, peakwidth, backgroundwidth, peakthresh )
%
% This calls nlProc_findSpectrumPeaks for every trial and channel waveform
% in a Field Trip dataset. This is intended to identify tone noise.
%
% "ftdata" is a ft_datatype_raw dataset.
% "peakwidth" is the relative width of the fine-resolution frequency bins
%   used for peak detection. A value of 0.1 would mean a bin width of 2 Hz
%   at a frequency of 20 Hz.
% "backgroundwidth" is the ratio between the upper and lower frequencies of
%   the span used to evaluate the spectrum background around putative peaks.
%   A value of 2.0 would mean evaluating noise over a one-octave span.
% "peakthresh" is the magnitude threshold for recognizing a peak in the
%   frequency spectrum. This is a multiple of the average local background.
%
% "peakfreqs" is a Ntrials x Nchannels cell array containing vectors with
%   detected peak frequencies.
% "peakheights" is a Ntrials x Nchannels cell array containing vectors with
%   detected peak heights normalized to the background level.
% "peakwidths" is a Ntrials x Nchannels cell array containing vectors with
%   detected relative peak widths (FWHM / frequency).
```

10.12 nlFT_getChannelCorrelMatrix.m

```
% function [ rmatrix pmatrix ] = ...
%   nlFT_getChannelCorrelMatrix( data_before, data_after, wantprogress )
%
% This checks for correlation between channels in the "before" and "after"
% datasets, returning the correlation coefficients and null hypothesis
% P-values for each (before,after) pair (per the "corrcoef" function).
%
% The datasets must have the same sampling rate, the same number of trials,
% and the same number of samples in corresponding trials. Trials are
% concatenated to compute correlation statistics.
%
% NOTE - This is slow! Time goes up as the square of the number of channels.
% A progress banner is printed.
%
% NOTE - Matrix values are NaN for cases where an input had zero variance.
%
% "data_before" is a Field Trip dataset prior to channel mapping.
% "data_after" is a Field Trip dataset after channel mapping.
% "wantprogress" is true to display a progress banner, false otherwise.
%   If omitted, it defaults to true.
%
% "rmatrix" is a matrix indexed by (chanbefore,chanafter) containing
%   correlation coefficient values.
% "pmatrix" is a matrix indexed by (chanbefore,chanafter) containing P-values
%   for the null hypothesis (that the channels are not correlated).
```

10.13 nlFT_getEndpointRamps.m

```
% function ramptrials = nlFT_getEndpointRamps( ftdata, spanfrac )
%
% This segments signals into NaN and non-NaN regions, and constructs
% piecewise-linear background connecting the endpoints of all non-NaN
% regions (defined across the entire data series, including NaN regions).
%
% This is intended to be subtracted before filtering to suppress ringing
% from discontinuities in the signal and its derivative.
%
% This is a wrapper for nlArt_getEndpointRamps.
%
% "ftdata" is a ft_datatype_raw dataset to process.
% "spanfrac" is the fraction of the length of non-NaN spans to keep at each
%   endpoint when estimating the line fit (e.g. 0.05 to pay attention to the
%   first and last 5% of each non-NaN segment). This is intended to suppress
%   transient samples at the endpoints. If this is 0 or NaN, only the first
%   and last sample of each non-NaN region are used.
```

```
%
% "ramptrials" is a copy of ftdata.trial with data replaced by piecewise
%   linear background fits.
```

10.14 nlFT_getEstimatedChannelMapping.m

```
% function [ srclabels, dstlabels ] = nlFT_getEstimatedChannelMapping( ...
%   data_before, data_after, rmatrix, pmatrix )
%
% This attempts to infer a channel mapping from "data_before" to "data_after"
% by looking at correlations between individual channels in each data set.
%
% The datasets must have the same sampling rate, the same number of trials,
% and the same number of samples in corresponding trials.
%
% Not all channels are guaranteed to be mapped. Channels that couldn't be
% mapped are omitted from the output. Order of labels in the output is not
% guaranteed.
%
% NOTE - Computing correlation values is slow! Time goes up as the square of
% the number of channels.
%
% "data_before" is a Field Trip dataset prior to channel mapping.
% "data_after" is a Field Trip dataset after channel mapping.
% "rmatrix" is the matrix of R-values returned by nlFT_getChannelCorrelMatrix.
%   If this argument is omitted, the R-value matrix is recomputed.
% "pmatrix" is the matrix of P-values returned by nlFT_getChannelCorrelMatrix.
%   If this argument is omitted, the P-value matrix is recomputed.
%
% "srclabels" is a cell array containing channel labels from "data_before".
% "dstlabels" is a cell array containing corresponding channel labels from
%   "data_after".
```

10.15 nlFT_getEventEdges.m

```
% function [ risetimes falltimes bothtimes ] = ...
%   nlFT_getEventEdges( ftevents, desiredlabel, samprate )
%
% This processes a Field Trip event list that is assumed to represent boolean
% (TTL) events and identifies the times of rising and falling edges.
%
% "ftevents" is a vector of Field Trip event records. Relevant fields are
%   "value", "sample", and "type" (containing the channel label).
% "desiredlabel" is a character vector containing the value to match to
%   events' "type" field. If this is empty, all events are accepted.
% "samprate" is the sampling rate (used to convert the "sample" field to
```

```
%   times in seconds). If this is NaN, the contents of "sample" are returned.
%
% "risetimes" contains timestamps (in seconds) of logical-1 events.
% "falltimes" contains timestamps (in seconds) of logical-0 events.
% "boththimes" contains timestamps (in seconds) of all matching events.
```

10.16 nlFT_getLabelChannelMapFromNumbers.m

```
% function [ maplabelsraw maplabelscooked ] = ...
%   nlFT_getLabelChannelMapFromNumbers( ...
%     chanmap, idxlabelsraw, idxlabelscooked )
%
% This function translates a channel mapping table defined using channel
% indices into a channel mapping table defined using labels.
%
% "chanmap" is a vector indexed by cooked channel number containing the raw
%   channel number that maps to each cooked location, or NaN if none does.
%   the first channel index is 1, per Matlab conventions.
% "idxlabelsraw" is a cell array containing all raw channel names.
% "idxlabelscooked" is a cell array containing all cooked channel names.
%
% "maplabelsraw" is a cell array containing raw channel names that correspond
%   to the names in "maplabelscooked".
% "maplabelscooked" is a cell array containing cooked channel names that
%   correspond to the names in "maplabelsraw".
```

10.17 nlFT_getMemChans.m

```
% function memchans = nlFT_getMemChans()
%
% This queries the maximum number of data channels that can be loaded into
% memory at one time (the "memchans" argument for NeuroLoop iterating
% functions).
%
% The higher this is, the faster data is read, due to not having to repeatedly
% scan over data files that store matrix data. The downside is that memory
% requirements can get big very quickly (typically 1 gigabyte per
% channel-hour of data).
%
% "memchans" is the current maximum number of memory-resident channels.
%
% FIXME - This stores state as global variables. This was the least-ugly
% way of passing tuning parameters to low-level reading functions.
```

10.18 nlFT_getNaNMask.m

```
% function trialmask = nlFT_getNaNMask( ftdata )
%
% This function builds a copy of a Field Trip "trial" cell array that contains
% logical matrices indicating which elements are NaN for each trial.
%
% This is intended to be used with "nlFT_fillNaN" and "nlFT_applyNaNMask".
% NaN segments are interpolated, filtering is performed, and then NaN
% segments are restored.
%
% "ftdata" is a ft_datatype_raw dataset to examine.
%
% "trialmask" is a 1xNtrials cell array containing NchansxNtime logical
%   matrices that are true for NaN elements in the original trial and false
%   otherwise.
```

10.19 nlFT_getTimelockAverage.m

```
% function chanaverage = nlFT_getTimelockAverage( thistimelock, avgchans )
%
% This computes the mean response across channels in thistimelock.avg.
%
% The average is computed using a subset of the channels.
%
% "thistimelock" is a dataset returned by ft_timelockanalysis.
% "avgchans" is a cell array containing channel labels to use for building
%   the average, or {} to use all channels.
%
% "chanaverage" is the average across channels in thistimelock.avg.
```

10.20 nlFT_getWantedChannels.m

```
% function [ typeswanted nameswanted bankswanted ] = nlFT_getWantedChannels()
%
% This reports the type, name, and bank patterns set by the most recent
% call to nlFT_selectChannels().
%
% Types must match exactly. Individual channel and bank names are valid
% regex patterns. Empty lists of patterns or labels always match, indicating
% absence of filtering on that particular criterion.
%
% "typeswanted" is a cell array containing a list of labels. The channel
%   type (from LoopUtil's "banktype" or Field Trip's "chantype") must be in
%   the list for the channel to be processed.
```

```
% "nameswanted" is a cell array containing a list of regex patterns. The
%   channel name (from Field Trip's "label") must match one of the regexes
%   in the list for the channel to be processed.
% "bankswanted" is a cell array containing a list of regex patterns. The
%   bank name (from LoopUtil; used as the prefix for channel names) must
%   match one of the regexes in the list for the channel to be processed.
%
% FIXME - This stores state as global variables. This was the least-ugly way
% of implementing channel and bank filtering without modifying Field Trip.
```

10.21 nlFT_getWindowsAroundEvents.m

```
% function masklist = ...
%   nlFT_getWindowsAroundEvents( ftdataset, window_ms, evtimes_sec )
%
% This function builds per-trial mask vectors selecting windows around
% trial triggering events (the t=0 point in trials), or against a
% user-specified list of event times.
%
% This handles overlapping trials and multiple events falling within a trial
% correctly.
%
% "ftdataset" is a structure of type ft_datatype_raw holding the trial data.
% "window_ms" [ start stop ] is a vector containing the time range to accept
%   around each event. This is in milliseconds.
% "evtimes_sec" is a vector containing event timestamps in seconds. If this
%   is [], the t=0 points in each trial are used instead.
%
% "masklist" is a 1xNtrials cell array with one cell per trial. Each cell
%   contains a 1xNsamples logical vector that's true in the accept window
%   and false elsewhere.
```

10.22 nlFT_guessMultipleExpDecays.m

```
% function fitlist = nlFT_guessMultipleExpDecays( ...
%   ftdata, fitconfig, want_plots, want_reports, plotconfig, ...
%   tattle_verbosity, report_verbosity )
%
% This is a wrapper for nlArt_guessMultipleExpDecays().
%
% This uses black magic to guess where exponential curve fits should be
% performed to remove artifacts. It can be robust but needs a lot of
% fine-tuning.
%
% Plots of curve fit attempts are optionally generated. These are useful
% when hand-tuning the configuration parameters.
```

```
%
% "ftdata" is a ft_datatype_raw structure with trial data to process.
% "fitconfig" is a configuration structure, per EXPGUESSCONFIG.txt.
% "want_plots" is true if debug plots of curve fits are to be generated.
% "want_reports" is true if summaries of curve fit progress are to be saved.
% "plotconfig" is a structure with the fields described in EXPGUESSPLOT.txt,
%   or struct([]) to suppress plotting and reports.
% "tattle_verbosity" is 'quiet', 'terse', 'normal', or 'verbose'. This
%   controls how many debugging/progress messages are sent to the console.
% "report_verbosity" is 'quiet', 'terse', 'normal', or 'verbose'. This
%   controls how many debugging/progress messages are written to disk.
%
% "fitlist" is a {ntrials, nchannels} cell array. Each cell array holds a
%   one-dimensional cell array containing curve fit parameters for
%   successive curve fits. Curve fit parameters are structures as described
%   in ARTFITPARAMS.txt.
```

10.23 nlFT_initReadTable.m

```
% function nlFT_initReadTable( datatable, chancolumns, timecolumn, ...
%   timefirst, timelast, samplespertimeunit, samprate )
%
% This stores a table and metadata to be read using nlFT_readTableHeader()
% and nlFT_readTableData(). The idea is to be able to give Field Trip a way
% to read non-uniformly-sampled tabular data as waveform data.
%
% FIXME - Support for nlFT_readTableEvents NYI.
%
% To release memory allocated for the copy of the table, call this with an
% empty table.
%
% "datatable" is the table to read.
% "chancolumns" is a cell array containing column labels of data columns.
% "timecolumn" is the label of the timestamp column.
% "timefirst" is the timestamp value corresponding to sample 1.
% "timelast" is the timestamp value corresponding to the last waveform sample.
% "samplespertimeunit" is the number of samples corresponding to a timestamp
%   change of 1.0 timestamp units. For seconds, this is equal to "samprate".
% "samprate" is the number of samples per second.
%
% FIXME - This stores state as global variables, including a copy of the
% table. This was the least-ugly way to store persistent state.
```

10.24 nlFT_iterateAcrossData.m

```
% function [ newtrials auxdata ] = nlFT_iterateAcrossData( ftdata, iterfunc )
```

```
%
% This iterates across the trials and channels in a Field Trip dataset,
% applying a processing function to each trial and channel's data. Processing
% output is aggregated and returned.
%
% "ftdata" is an "ft_datatype_raw" data structure.
% "iterfunc" is a function handle used to transform channel waveform data
%   into "result" data, per FT_ITERFUNC.txt.
%
% "newtrials" is a processed copy of the "trial" field, containing modified
%   per-trial and per-channel waveform data.
% "auxdata" is a cell array indexed by {trial,channel} containing auxiliary
%   data returned by the iteration processing function.
```

10.25 nlFT_iterateAcrossFolderBatching.m

```
% function auxdata = nlFT_iterateAcrossFolderBatching( ...
%   config_load, iterfunc_batched, chanbatchsize, trialbatchsize, verbosity )
%
% This iterates across a Field Trip dataset, loading a few channels at a
% time and/or a few trials at a time and applying a processing function to
% each channel subset. Processing output is aggregated and returned.
%
% The idea is to be able to process a dataset much larger than can fit in
% memory. At 30 ksps the footprint is typically about 1 GB per channel-hour.
%
% This calls an iteration function handle of the type described by
% FT_ITERFUNC_BATCHED.txt.
%
% NOTE - If the iteration function returns anything for "ftdata_new", make
% sure the aggregated result is small enough to fit in memory!
%
% "config_load" is a Field Trip configuration structure to be passed to
%   ft_preprocessing() to load the data. The "channel" field is split into
%   batches when iterating.
% "iterfunc_batched" is a function handle used to transform channel waveform
%   data into "result" data, per FT_ITERFUNC_BATCHED.txt.
% "chanbatchsize" is the number of channels to process at a time. Set this to
%   inf to process all channels at once.
% "trialbatchsize" is the number of trials to process at a time. Set this to
%   inf to process all trials at once.
% "verbosity" is an optional argument. It can be set to 'none' (no console
%   output), 'terse' (reporting the number of channels processed), or 'full'
%   (reporting the names of channels processed). The default is 'none'.
%
% "ftdata" is a "ft_datatype_raw" structure built by aggregating the Field
%   Trip data structures ("ftdata_new") returned by the iteration processing
%   function. These are assumed to be compatible (same sampling rate, etc),
```

```
%   and the aggregated result must fit in memory. If the iteration function
%   returns struct([]) as ftdata_new, "ftdata" is also set to struct([]).
% "auxdata" is a cell array indexed by {trial,channel} containing the
%   auxiliary data ("auxdata") returned by the iteration processing
%   function.
```

10.26 nlFT_makeEmptyEventList.m

```
% function evlist = nlFT_makeEmptyEventList()
%
% This function makes a struct array of length zero with the fields that
% may be returned by ft_read_event().
%
% The idea is to provide a structure that works with both "isempty(evlist)"
% and "foo = [ evlist(:).bar ]".
%
% No arguments.
%
% "evlist" is a struct array of length zero with the required and optional
%   fields.
```

10.27 nlFT_makeFTDataFromMatrices.m

```
% function ftdata = nlFT_makeFTDataFromMatrices( ...
%   wavedata, samprate, trigoffsetsamps, trigtimes, labelprefix )
%
% This builds a ft_datatype_raw structure containing supplied waveform data.
%
% "wavedata" is either a Nchans x Nsamples x Ntrials matrix or a cell array
%   with Ntrials cells, each containing a Nchans x Nsamples matrix.
% "samprate" is the sampling rate.
% "trigoffsetsamps" specifies when the trigger time is within each trial.
%   This is 0 if the first sample in the trial is at the trigger, positive
%   if a later sample in the trial is the trigger, and negative if the
%   trigger occurred before the first sample in the trial.
% "trigtimes" is a vector with trigger times for each trial. This is used for
%   constructign trial definitions. If necessary, an offset is added to
%   guarantee that all samples have positive indices in the global data.
%   Specify [] to automatically build trigger times for trial definitions.
% "labelprefix" is a character vector containing a prefix to use when
%   constructing channel labels.
%
% "ftdata" is a ft_datatype_raw structure containing trial data. This
%   includes a header and a config structure with cfg.trl.
```

10.28 nlFT_makeFTName.m

```
% function newlabel = nlFT_makeFTName( banklabel, channum )
%
% This turns a NeuroLoop bank label and channel number into a Field Trip
% channel label.
%
% "banklabel" is the NeuroLoop bank label (a valid field name character array).
% "channum" is the NeuroLoop channel number (an arbitrary nonnegative integer).
%
% "newlabel" is a character array containing the corresponding Field Trip
%   channel label.
```

10.29 nlFT_makeLabelsFromNumbers.m

```
% function chanlabels = nlFT_makeLabelsFromNumbers( bankname, channums )
%
% This function calls "nlFT_makeFTName" to convert channel numbers into
% labels for a list of channel numbers.
%
% "bankname" is the bank name to use when building channel labels.
% "channums" is a vector containing channel numbers.
%
% "chanlabels" is a Nx1 cell array containing channel labels.
```

10.30 nlFT_mapChannelLabels.m

```
% function newlabels = nlFT_mapChannelLabels( oldlabels, lutold, lutnew )
%
% This function translates channel labels according to a lookup table.
% This is intended to be used for channel mapping.
%
% "oldlabels" is a cell array containing labels to translate.
% "lutold" is a cell array containing old labels that correspond to the
%   labels in "lutnew".
% "lutnew" is a cell array containing new labels that correspond to the
%   labels in "lutold".
%
% "newlabels" is a cell array of translated labels. Old labels are turned
%   into corresponding new labels. Labels that can't be translated are
%   replaced with ''.
```

10.31 nlFT_mergeEvents.m

```
% function eventsmerged = nlFT_mergeEvents( alleventmeta )
%
% This serializes the multiple event lists produced by nlFT_readAllEvents(),
% merging them into a single Field Trip event list.
%
% This discards the event metadata, and replaces the "type" field of each
% event with the Field Trip channel label for the event's source.
%
% "alleventmeta" is a vector of structures containing event channel metadata
%   and Field Trip event lists for each channel, per nlFT_readAllEvents().
%
% "eventsmerged" is a Field Trip event list struct vector containing all
%   events. Stored fields are:
%   "sample" - Event onset time in samples.
%   "value" - Event value (boolean or integer code).
%   "type" - Character array with the event source's channel label.
```

10.32 nlFT_parseChannelsIntoBanks.m

```
% function bankdata = nlFT_parseChannelsIntoBanks( chanlabels, wavedata )
%
% This tries to identify channels' banks by parsing channel labels, and
% optionally divides one trial's data into per-bank data. This will only
% work with channel labels generated by the LoopUtil library (i.e. with
% data read using the LoopUtil Field Trip I/O functions).
%
% "chanlabels" is a cell array containing channel names. This is normally
%   taken from header.label or rawdata.label.
% "wavedata" is a Nchans*Nsamples matrix containing waveform data. This is
%   normally taken from rawdata.trial{k}. An empty matrix skips data copying.
%
% "bankdata" is a structure indexed by LoopUtil bank identifier. Each field
%   contains a bank data structure with a "label" field containing a cell
%   array of FT channel labels, a "channum" field containing a vector of
%   LoopUtil channel indices, and optionally a "wavedata" field containing
%   a Nchans*Nsamples matrix of waveform data.
```

10.33 nlFT_parseFTName.m

```
% function [ bankid chanid ] = nlFT_parseFTName( chanlabel )
%
% This parses a name generated by nlFT_makeFTName, turning the compound name
% back into bank and channel IDs.
```

10.34 nlFT_pruneFTEvents.m

```
% function newlist = nlFT_pruneFTEvents(oldlist)
%
% This traverses a list of events stored in Field Trip format, removing
% event records which, for a given "type", have the same "value" as the
% previously-seen event of that "type".
%
% This is intended to be used with event lists generated by the LoopUtil
% library, which store TTL state changes as events with "type" holding the
% channel label.
%
% "oldlist" is the Field Trip event list to process.
%
% "newlist" is the pruned list.
```

10.35 nlFT_rampOverStimStep.m

```
% function newftdata = ...
%   nlFT_rampOverStimStep( oldftdata, ramp_span, stim_span )
%
% This is a wrapper for nlArt_rampOverStimStep().
%
% This NaNs out a stimulation artifact region and applies a ramp over a
% larger portion of each trial to turn the stepwise level shift across the
% stimulation region into a more graceful change (to avoid filter ringing
% in subsequent processing steps).
%
% This is intended to be used after artifact cancellation, so that there
% aren't large excursions in the post-stimulation signal.
%
% "oldftdata" is a ft_datatype_raw structure with trial data to process.
% "ramp_span" [ min max ] is a time range over which to apply the ramp.
% "stim_span" [ min max ] is a time range containing stimulation artifacts
%   to be squashed, or [] to auto-detect existing NaN spans (which may
%   be different lengths for each trial/channel).
%
% "newftdata" is a copy of "oldftdata" with stimation regions in each trial
%   NaNed out and a gradual ramp between pre-stimulation and
%   post-stimulation DC levels.
```

10.36 nlFT_readAllEvents.m

```
% function alleventmeta = nlFT_readAllEvents( indir, wantpromote )
%
```

```
% This probes the specified directory using nlIO_readFolderMetadata(), and
% calls nlFT_readEvents() for any channels of type "eventbool" or
% "eventwords" that pass the nlFT_selectChannels() criteria. The results for
% each channel are returned, with metadata.
%
% NOTE - Field Trip channel number is sensitive to filtering. Use
% nlFT_findChannelIndices() to modify ftchanidx if filtering changes.
%
% "indir" is the directory to process.
% "wantpromote" is true if continuous data ("integer", "bool", and
%   "flagvector" types) is to be converted into sparse data, and false if not.
%
% "alleventmeta" is a vector of structures with the following fields:
%   "ftchanlabel" is the Field Trip channel label for this channel.
%   "ftchanidx" is the Field Trip channel number for this channel.
%   "ftchantype" is the Field Trip channel type for this channel.
%   "nlbankid" is the NeuroLoop bank name for this channel.
%   "nlchanid" is the NeuroLoop channel number for this channel.
%   "ftevents" is the Field Trip event list struct array for this channel.
```

10.37 nlFT_readDataDouble.m

```
% function data = ...
%   nlFT_readDataDouble( indir, header, firstsample, lastsample, chanidxlist )
%
% This probes the specified directory using nlIO_readFolderMetadata(), and
% reads all appropriate signal data into a Field Trip data matrix.
%
% This is intended to be called by ft_read_data() via the "dataformat"
% argument.
%
% The "Double" version of this function promotes data to double-precision
% floating-point.
%
% NOTE - This returns monolithic data (a 2D matrix), not epoched data.
% NOTE - This requires all selected banks to have the same sampling rate and
% number of samples!
%
% This calls nlFT_testWantChannel() and nlFT_testWantBank() and only reads
% channels that are wanted. By default all channels and banks are wanted;
% use nlFT_selectChannels() to change this.
%
% If directory probing fails, and error is thrown.
%
% "indir" is the directory to process.
% "header" is the Field Trip header associated with this directory.
% "firstsample" is the index of the first sample to read.
% "lastsample" is the index of the last sample to read.
```

```
% "chanidxlist" is a vector containing Field Trip channel indices to read.
%
% "data" is the resulting 2D data matrix.
```

10.38 nlFT_readDataNative.m

```
% function data = ...
%   nlFT_readDataNative( indir, header, firstsample, lastsample, chanidxlist )
%
% This probes the specified directory using nlIO_readFolderMetadata(), and
% reads all appropriate signal data into a Field Trip data matrix.
%
% This is intended to be called by ft_read_data() via the "dataformat"
% argument.
%
% The "Native" version of this function keeps data as its native type.
%
% NOTE - This returns monolithic data (a 2D matrix), not epoched data.
% NOTE - This requires all selected banks to have the same sampling rate and
% number of samples!
% NOTE - This requires all selected banks to have compatible types! The first
% channel processed determines the returned type. For best results, make sure
% all selected banks have the _same_ type.
%
% This calls nlFT_testWantChannel() and nlFT_testWantBank() and only reads
% channels that are wanted. By default all channels and banks are wanted;
% use nlFT_selectChannels() to change this.
%
% If directory probing fails, and error is thrown.
%
% "indir" is the directory to process.
% "header" is the Field Trip header associated with this directory.
% "firstsample" is the index of the first sample to read.
% "lastsample" is the index of the last sample to read.
% "chanidxlist" is a vector containing Field Trip channel indices to read.
%
% "data" is the resulting 2D data matrix.
```

10.39 nlFT_readData_helper.m

```
% function data = nlFT_readData_helper( indir, wantnative, ...
%   header, firstsample, lastsample, chanidxlist )
%
% This probes the specified directory using nlIO_readFolderMetadata(), and
% reads all appropriate signal data into a Field Trip data matrix.
%
```

```
% This may either promote data to double or keep it as its native type,
% depending on "wantnative".
%
% NOTE - This returns monolithic data (a 2D matrix), not epoched data.
% NOTE - This requires all selected banks to have the same sampling rate and
% number of samples!
%
% This calls nlFT_testWantChannel() and nlFT_testWantBank() and only saves
% channels that are wanted. By default all channels and banks are wanted;
% use nlFT_selectChannels() to change this.
%
% If directory probing fails, an error is thrown.
%
% "indir" is the directory to process.
% "wantnative" is true to store native-format data and false to promote to
%   double-precision floating-point.
% "header" is the Field Trip header associated with this directory.
% "firstsample" is the index of the first sample to read (starting at 1).
% "lastsample" is the index of the last sample to read.
% "chanidxlist" is a vector containing Field Trip channel indices to read.
%
% "data" is the resulting 2D data matrix.
```

10.40 nlFT_readEvents.m

```
% function eventlist = nlFT_readEvents( indir, header )
%
% This probes the specified directory using nlIO_readFolderMetadata(), and
% reads events from all sparse (event-type) channels found. The channel name
% for each event is stored in the event's "type" field.
%
% This is intended to be called by ft_read_event() via the "eventformat"
% argument.
%
% NOTE - Field Trip expects this to return the header, rather than an event
% list, if it's called with just one argument ("indir").
%
% NOTE - Timestamps are guaranteed to be in order, but there are no order
% guarantees for events with the same timestamp (such as simultaneous events
% from different channels).
%
% "indir" is the directory to process.
% "header" is the Field Trip header associated with this directory.
%
% "eventlist" is a vector of field trip event records with the "sample",
%   "value", and "type" fields filled in. The "type" field contains the
%   label of the channel that sourced the event.
```

10.41 nlFT_readEventsContinuous.m

```
% function eventlist = nlFT_readEventsContinuous( indir, header )
%
% This probes the specified directory using nlIO_readFolderMetadata(), and
% reads events from all sparse (event-type) or discrete-valued continuous
% (boolean/integer/flagvector) channels found. The channel name for each
% event is stored in the event's "type" field.
%
% This is intended to be called by ft_read_event() via the "eventformat"
% argument.
%
% NOTE - Field Trip expects this to return the header, rather than an event
% list, if it's called with just one argument ("indir").
%
% NOTE - Timestamps are guaranteed to be in order, but there are no order
% guarantees for events with the same timestamp (such as simultaneous events
% from different channels).
%
% "indir" is the directory to process.
% "header" is the Field Trip header associated with this directory.
%
% "eventlist" is a vector of field trip event records with the "sample",
%   "value", and "type" fields filled in. The "type" field contains the
%   label of the channel that sourced the event.
```

10.42 nlFT_readHeader.m

```
% function header = nlFT_readHeader(indir)
%
% This probes the specified directory using nlIO_readFolderMetadata(),
% and translates the folder's metadata into a Field Trip header.
%
% This is intended to be called by ft_read_header() via the "headerformat"
% argument.
%
% This calls nlFT_testWantChannel() and nlFT_testWantBank() and only saves
% channels that are wanted. By default all channels and banks are wanted;
% use nlFT_selectChannels() to change this.
%
% NOTE - The following nonstandard fields are added to the header. All of
% these are cell arrays with per-channel metadata copied from the LoopUtil
% bank metadata (per FOLDERMETA.txt).
%   "channativezero" is the native data value representing a signal value of
%     zero.
%   "channativescale" is a multiplier used to convert "native" data values to
%     double-precision floating-point data values in appropriate units.
```

```
%   "chanflagbits" is a structure indexed by flag label containing the
%     integer bit-mask values that correspond to each flag, for "flagvector"
%     data. This is an empty structure for other data.
%
% If probing fails, an error is thrown.
%
% "indir" is the directory to process.
%
% "header" is the resulting Field Trip header.
```

10.43 nlFT_readTableData.m

```
% function data = ...
%   nlFT_readTableData( fname, header, firstsample, lastsample, chanidxlist )
%
% This constructs Field Trip waveform data based on information previously
% supplied to nlFT_initReadTable(). The idea is to be able to give Field
% Trip a way to read non-uniformly-sampled tabular data as waveform data.
%
% This is intended to be called by ft_read_data() via the "dataformat"
% argument.
%
% NOTE - This returns monolithic data (a 2D matrix), not epoched data.
%
% "fname" is the filename passed to ft_read_data(). This is ignored.
% "header" is the Field Trip header returned by nlFT_readTableHeader().
% "firstsample" is the index of the first sample to read.
% "lastsample" is the index of the last sample to read.
% "chanidxlist" is a vector containing Field Trip channel indices to read.
%
% "data" is the resulting 2D data matrix.
```

10.44 nlFT_readTableHeader.m

```
% function header = nlFT_readTableHeader(fname)
%
% This constructs a Field Trip header based on information previously
% supplied to nlFT_initReadTable(). The idea is to be able to give Field
% Trip a way to read non-uniformly-sampled tabular data as waveform data.
%
% This is intended to be called by ft_read_header() via the "headerformat"
% argument.
%
% "fname" is the filename passed to ft_read_header(). This is ignored.
```

10.45 nlFT_removeArtifactsSigma.m

```
% function newftdata = nlFT_removeArtifactsSigma( oldftdata, ...
%   ampthresh, derivthresh, ampthreshfall, derivthreshfall, ...
%   trimbefore_ms, trimafter_ms, smoothsecs, dcsecs )
%
% This identifies artifacts as excursions in signals' 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.
%
% This is a wrapper for nlArt_removeArtifactsSigma.
%
% "oldftdata" is a Field Trip dataset 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_ms" is the number of milliseconds to squash before the artifact.
% "trimafter_ms" is the number of milliseconds to squash after the artifact.
% "derivsmooth_ms" is the size in milliseconds of the smoothing window to
%   apply before taking the derivative, or 0 or NaN for no smoothing.
% "dcsecs" is the size in seconds of the window for computing local DC
%   average removal ahead of computing statistics.
%
% "newftdata" is a copy of "oldftdata" with artifacts replaced with NaN.
```

10.46 nlFT_removeMultipleExpDecays.m

```
% function [ newftdata fitlist ] = ...
%   nlFT_removeMultipleExpDecays( oldftdata, fenceposts, method )
%
% This is a wrapper for nlArt_removeMultipleExpDecays().
%
% This attempts to curve fit and remove settling artifacts composed of
% multiple exponential decay tails.
%
% DC levels are discarded from the curve fits - the "offset" parameters in
% the curve fits are all set to zero.
%
% "oldftdata" is a ft_datatype_raw structure with trial data to process.
% "fenceposts" is a vector containing times to be used as span endpoints for
%   curve fitting. The span between the last two times is curve fit and its
%   contribution subtracted, then the next-last span, and so forth. Only the
%   portion of the wave following the earliest fencepost is modified.
% "method", if present, is a character vector or cell array specifying the
%   algorithms to use to perform exponential fits ('log', 'pinmax', or
```

```
%   'pinboth', per "nlArt_fitExpDecay()"). If this is a character vector,
%   the same algorithm is used for all fits. If this is a cell array, it
%   should have one fewer elements than "fenceposts", and specifies the
%   algorithm used for each fit. If "method" is absent, or if any method is
%   specified as '', a default method is chosen.
%
% "newftdata" is a copy of "oldftdata" with the curve fits subtracted.
% "fitlist" is a {ntrials, nchannels, nfits} cell array holding curve fit
%   parameters for successive curve fits (farthest/slowest first). Curve fit
%   parameters are structures as described in ARTFITPARAMS.txt.
```

10.47 nlFT_selectAllChannels.m

```
% function nlFT_selectAllChannels()
%
% This clears LoopUtil's channel filtering.
% This is a wrapper for nlFT_selectChannels().
```

10.48 nlFT_selectChannels.m

```
% function nlFT_selectChannels( typeswanted, nameswanted, bankswanted )
%
% This sets conditions that chanels must match in order to be processed.
% Channels must have a desired type, have a name that matches a desired
% regex, or be part of a bank whose label matches a desired regex.
%
% Individual channel and bank names are valid regex patterns.
%
% Empty lists always match (so pass "{}" as a condition to disable that test).
%
% "typeswanted" is a cell array containing a list of labels. The channel
%   type (from LoopUtil's "banktype" or Field Trip's "chantype") must be in
%   the list for the channel to be processed.
% "nameswanted" is a cell array containing a list of regex patterns. The
%   channel name (from Field Trip's "label") must match one of the regexes
%   in the list for the channel to be processed.
% "bankswanted" is a cell array containing a list of regex patterns. The
%   bank name (from LoopUtil; used as the prefix for channel names) must
%   match one of the regexes in the list for the channel to be processed.
%
% FIXME - This stores state as global variables. This was the least-ugly way
% of implementing channel and bank filtering without modifying Field Trip.
```

10.49 nlFT_selectOneFTChannel.m

```
% function nlFT_selectOneFTChannel( chanlabel )
%
% This sets up LoopUtil's channel filtering to pass one specific channel
% using Field Trip's channel label.
%
% This is a wrapper for nlFT_selectChannels().
%
% "chanlabel" is the Field Trip channel label for the desired channel.
```

10.50 nlFT_selectTrials.m

```
% function newftdata = nlFT_selectTrials( oldftdata, trialmask )
%
% This keeps only the specified subset of trials in a Field Trip data
% structure, adjusting structure fields appropriately.
%
% NOTE - Field Trip will complain vigorously if you give it a data structure
% with no trials! Only call this if at least one trial will be kept.
%
% "oldftdata" is the Field Trip data structure to modify.
% "trialmask" is a logical vector with one entry per trial, which is true
%   for the trials to be kept and false for the trials to discard.
%
% "newftdata" is a copy of "oldftdata" containing only the desired trials.
```

10.51 nlFT_setMemChans.m

```
% function nlFT_setMemChans(newcount)
%
% This sets the maximum number of data channels that can be loaded into
% memory at one time (the "memchans" argument for NeuroLoop iterating
% functions).
%
% The higher this is, the faster data is read, due to not having to repeatedly
% scan over data files that store matrix data. The downside is that memory
% requirements can get big very quickly (typically 1 gigabyte per
% channel-hour of data).
%
% "newcount" is the new maximum number of memory-resident channels.
%
% FIXME - This stores state as global variables. This was the least-ugly
% way of passing tuning parameters to low-level reading functions.
```

10.52 nlFT_subtractCurveFits.m

```
% function newftdata = nlFT_subtractCurveFits( ..
%   oldftdata, timerange, fitlist, dcflag )
%
% This reconstructs and removes a series of curve fits from each trial and
% channel in a Field Trip dataset.
%
% "oldftdata" is a ft_datatype_raw structure with trial data to modify.
% "timerange" [ min max ] is a time span over which to reconstruct the
%   curve fits, in seconds.
% "fitlist" is a {ntrials, nchannels} cell array. Each cell array holds a
%   one-dimensional cell array containing curve fit parameters for zero or
%   more curve fits. Curve fit parameters are structures as described in
%   ARTFITPARAMS.txt.
% "dcflag" is 'keepdc' or 'ignoredc', indicating whether to keep or strip
%   the DC component of the curve fits.
%
% "newftdata" is a copy of "oldftdata" with curve fits subtracted.
```

10.53 nlFT_subtractTimelockBackground.m

```
% function newtimelock = nlFT_subtractTimelockBackground( oldtimelock, bgvec )
%
% This subtracts a common background vector from each channel in
% oldtimelock.avg. The idea is to make local changes in response more visible.
%
% The background may be the mean (computed via nlFT_getTimelockAverage) or
% may be computed by other methods.
%
% "oldtimelock" is a dataset returned by ft_timelockanalysis.
% "bgvec" is a background vector to subtract from all channels.
%
% "newtimelock" is a copy of "oldtimelock" with the background vector
%   subtracted from newtimelock.avg.
```

10.54 nlFT_sumTrialArrays.m

```
% function newtrials = nlFT_sumTrialArrays( ...
%   firsttrials, firstweight, secondtrials, secondweight )
%
% This walks through two "trial" cell arrays (from ft_datatype_raw) and
% computes a weighted sum of correspnding trial matrices. Corresponding
% trial matrices must have the same dimensions.
%
```

```
% Among other things, this is intended to be used to add or subtract
% curve fits or background fits from trial data. For example:
%
%  foodata.trial = nlFT_sumTrialArrays(foodata.trial, 1, bgtrials, -1);
%
% "firsttrials" is a cell array containing trial matrices.
% "firstweight" is the weight to apply to trials in "firstrials".
% "secondtrials" is a cell array containing trial matrices.
% "secondweight" is the weight to apply to trials in "secondtrials".
%
% "newtrials" is a cell array containing trial matrices that are a weighted
%   sum of the trials in "firsttrials" and "secondtrials".
```

10.55 nlFT_testWantBank.m

```
% function iswanted = nlFT_testWantBank(bankname, banktype)
%
% This tests to see if a given bank is desired, per nlFT_selectChannels().
% A bank is desired if its type and bank name are both acceptable.
%
% "bankname" is the bank name to test.
% "banktype" is the bank type label to test.
%
% FIXME - This stores state as global variables. This was the least-ugly way
% of implementing channel and bank filtering without modifying Field Trip.
```

10.56 nlFT_testWantChannel.m

```
% function iswanted = nlFT_testWantChannel(channelname)
%
% This tests to see if a given channel name is in the list of desired channel
% names, per nlFT_selectChannels().
%
% "channelname" is the channel name to test.
%
% FIXME - This stores state as global variables. This was the least-ugly way
% of implementing channel and bank filtering without modifying Field Trip.
```

10.57 nlFT_uncompressFTEvents.m

```
% function evstructarray = nlFT_uncompressFTEvents(evtable, typelut)
%
% This de-compresses a compressed Field Trip event list that was produced by
```

```
% nlFT_compressFTEvents(). Table columns are converted back into structure
% fields, the "type" column is converted back to cell data, and empty
% "offset" and "duration" fields are created if not already present.
%
% If an empty LUT is given, the "type" column is copied as-is rather than
% translated. Otherwise all compressed "type" values must be valid indices
% into the lookup table.
%
% "evtable" is a table containing compressed event data.
% "typelut" is a cell array containing values to store in the "type" field
%   in the reconstructed structure array.
%
% "evstructarray" is a Field Trip event list (structure array).
```

Chapter 11

"nlIntan" Functions

11.1 nlIntan_iterateFolderChannels.m

```
% function folderresults = nlIntan_iterateFolderChannels( ...
%   foldermetadata, folderchanlist, memchans, procfunc, procmeta, procfid )
%
% This processes a folder containing Intan-format data, iterating through
% a list of channels, loading each channel's waveform data in sequence and
% calling a processing function with that data. Processing output is
% aggregated and returned.
%
% This is implemented such that only a few channels are loaded at a time.
%
% "Native" channel time series are stored as sample numbers (not times).
% "Cooked" channel time series are in seconds. Cooked analog data is
% converted to the units specified in the bank metadata (volts or microvolts).
% Cooked TTL data is converted to boolean.
%
% "foldermetadata" is a folder-level metadata structure, per FOLDERMETA.txt.
% "folderchanlist" is a structure listing channels to process; it is a
%   folder-level channel list per CHANLIST.txt.
% "memchans" is the maximum number of channels that may be loaded into
%   memory at the same time.
% "procfunc" is a function handle used to transform channel waveform data
%   into "result" data, per PROCFUNC.txt.
% "procmeta" is the object to pass as the "metadata" argument of "procfunc".
% "procfid" is the label to pass as the "folderid" argument of "procfunc".
%
% "folderresults" is a folder-level channel list structure that has
%   bank-level channel lists augmented with a "resultlist" field, per
%   CHANLIST.txt. The "resultlist" field is a cell array containing
%   per-channel output from "procfunc".
```

11.2 nlIntan_probeFolder.m

```
% function foldermeta = nlIntan_probeFolder( indir )
%
% This checks for the existence of Intan-format data files in the specified
% folder, and constructs a folder metadata structure if data is found.
%
% If no data is found, an empty structure is returned.
%
% "indir" is the directory to search.
%
% "foldermeta" is a folder metadata structure, per FOLDERMETA.txt.
```

Chapter 12

"vIntan" Functions

12.1 vIntan_readHeader.m

```
% function metadata = vIntan_readHeader( fname )
%
% This reads the metadata header from Intan ".rhd" and ".rhs" files.
% If reading fails, an empty structure array is returned.
%
% This file is derived from code supplied by Intan Technologies (used and
% re-licensed with permission).
%
% "fname" is the name of the file to read from, including path.
%
% "metadata" is a structure containing header data, per "INTAN_METADATA.txt".
```

Chapter 13

"nlOpenE" Functions

13.1 nlOpenE_assembleWords.m

```
% function wordvals = nlOpenE_assembleWords(bytevals, wordtype)
%
% This function assembles the bytes stored in an event list's "FullWords"
% array into word values. This tolerates a list with zero events.
%
% "bytevals" is a copy of the FullWords array (Nevents x Nbytes uint8 LE).
% "wordtype" is a character array containing the name of the type to promote
%   to (typically 'uint16', 'uint32', or 'uint64').
%
% "wordvals" is a vector containing assembled word values.
```

13.2 nlOpenE_getCrossingDetectThresholdDesc_v5.m

```
% function [ descsummary descdetailed ] = ...
%   nlOpenE_getCrossingDetectThresholdDesc_v5( procmeta )
%
% This generates a human-readable description of the threshold configuration
% for a TNE Lab crossing detector plugin.
%
% "procmeta" is the processor node metadata, per PROCMETA_OPENEPHYSv5.txt.
%
% "descsummary" is a cell array containing character vectors that are
%   individual lines of a human-readable summary of the configuration.
% "descdetailed" is a cell array containing character vectors that are
%   individual lines of a human-readable detailed description of the
%   configuration.
```

13.3 nlOpenE_getIntanRecorderSamprateFromIndex.m

```
% function samprate = nlOpenE_getIntanRecorderSamprateFromIndex( rateidx )
%
% This translates the Open Ephys configuration selection value for sampling
% rate into an actual sampling rate.
%
% This is defined by the order in which they're added in the
% SampleRateInterface object's combobox in RHD2000Editor.cpp.
%
% "rateidx" is the sampling rate index (should be 1..17).
%
% "samprate" is the sampling rate in Hz.
```

13.4 nlOpenE_getSettingsFileFromDataFolder_v5.m

```
% function settingsname = ...
%   nlOpenE_getSettingsFileFromDataFolder_v5( foldername )
%
% This parses an Open Ephys experiment folder path (the folder containing
% "structure.oebin"), and infers the path and filename of the corresponding
% "settings.xml" or "settingsN.xml" file.
%
% In Open Ephys v0.5, the "structure.oebin" file is in a folder named
% "experimentX/recordingY/", and the settings files are in the same folder
% that contains the "experimentX" subfolders. For "experiment1", the settings
% file is named "settings.xml". For other "experimentX" folders, the settings
% files are named "settings_X.xml".
%
% "foldername" is the full path of the folder containing "structure.oebin".
%
% "settingsname" is the full path of the appropriate settings XML file.
%   If parsing failed, this is an empty character array.
```

13.5 nlOpenE_iterateFolderChannels.m

```
% function folderresults = nlOpenE_iterateFolderChannels( ...
%   foldermetadata, folderchanlist, memchans, procfunc, procmeta, procfid )
%
% This processes a folder containing Open Ephys format data, iterating
% through a list of channels, loading each channel's waveform data in
% sequence and calling a processing function with that data. Processing
% output is aggregated and returned.
%
% This is implemented such that only a few channels are loaded at a time.
```

```
%
% "Native" channel time series are stored as sample numbers (not times).
% "Cooked" channel time series are in seconds. Cooked analog data is
% converted to the units specified in the bank metadata (volts or microvolts).
% Cooked TTL data is converted to boolean.
%
% "foldermetadata" is a folder-level metadata structure, per FOLDERMETA.txt.
% "folderchanlist" is a structure listing channels to process; it is a
%   folder-level channel list per CHANLIST.txt.
% "memchans" is the maximum number of channels that may be loaded into
%   memory at the same time.
% "procfunc" is a function handle used to transform channel waveform data
%   into "result" data, per PROCFUNC.txt.
% "procmeta" is the object to pass as the "metadata" argument of "procfunc".
% "procfid" is the label to pass as the "folderid" argument of "procfunc".
%
% "folderresults" is a folder-level channel list structure that has
%   bank-level channel lists augmented with a "resultlist" field, per
%   CHANLIST.txt. The "resultlist" field is a cell array containing
%   per-channel output from "procfunc".
```

13.6 nlOpenE_parseChannelMapGeneric_v5.m

```
% function thismap = ...
%   nlOpenE_parseChannelMapGeneric_v5( chanlist, reflist, reflut, enlist )
%
% This parses an array of source channel indices indices, an array of
% reference set indices, a reference lookup list, and an array of "enabled"
% flags, and assembles a structure describing this channel mapping (per
% "OPENEPHYS_CHANMAP.txt").
%
% NOTE - Reference banks start at 1, not 0. Convert before calling.
%
% This is intended to be called by other nlOpenE_parseChannelMap functions.
%
% "chanlist" is a vector indexed by new channel number containing the old
%   channel number that maps to each new location, or NaN if none does.
% "reflist" is a vector indexed by new channel number containing the
%   reference bank number to be used with each new channel. This may be [].
% "reflut" is a vector indexed by reference bank number listing the old
%   channel number to be used as a reference for each reference bank.
%   This may be [].
% "enlist" is a vector of boolean values indexed by new channel number
%   indicating which new channels are enabled. This may be [].
%
% "thismap" is a structure with the following fields:
%   "oldchan" is a vector indexed by new channel number containing the old
%     channel number that maps to each new location, or NaN if none does.
```

```
%   "oldref" is a vector indexed by new channel number containing the old
%     channel number to be used as a reference for each new location, or
%     NaN if unspecified.
%   "isenabled" is a vector of boolean values indexed by new channel number
%     indicating which new channels are enabled.
```

13.7 nlOpenE_parseChannelMapJSON_v5.m

```
% function maplist = nlOpenE_parseChannelMapJSONv5( jsonstruct )
%
% This parses a structure containing decoded JSON describing the
% configuration of an Open Ephys version 0.5.x channel map node.
%
% NOTE - Open Ephys labels streams in the channel map starting at 0.
% Matlab will index them in the struct array starting at 1.
%
% "jsonstruct" is a structure containing JSON data, from "jsondecode()".
%
% "maplist" is a structure array with one entry per mapping table found in
%   the original structure. The fields (per "OPENEPHYS_CHANMAP.txt") are:
%   "oldchan" is a vector indexed by new channel number containing the old
%     channel number that maps to each new location, or NaN if none does.
%   "oldref" is a vector indexed by new channel number containing the old
%     channel number to be used as a reference for each new location, or
%     NaN if unspecified.
%   "isenabled" is a vector of boolean values indexed by new channel number
%     indicating which new channels are enabled.
```

13.8 nlOpenE_parseChannelMapXML_v5.m

```
% function maplist = nlOpenE_parseChannelMapXML_v5( xmlstruct )
%
% This parses a structure containing a decoded Open Ephys version 0.5.x
% configuration file, and collects any channel mapping information it can
% find.
%
% NOTE - We're listing channel maps in the order that we find them, not
% sorted by stream number.
%
% "xmlstruct" is a structure containing XML configuration data, as read by
%   "readstruct()".
%
% "maplist" is a structure array with one entry per mapping table found in
%   the configuration file. The fields (per "OPENEPHYS_CHANMAP.txt") are:
%   "oldchan" is a vector indexed by new channel number containing the old
%     channel number that maps to each new location, or NaN if none does.
```

```
%   "oldref" is a vector indexed by new channel number containing the old
%     channel number to be used as a reference for each new location, or
%     NaN if unspecified.
%   "isenabled" is a vector of boolean values indexed by new channel number
%     indicating which new channels are enabled.
```

13.9 nlOpenE_parseConfigAudioBufferInfo_v5.m

```
% function audiometa = nlOpenE_parseConfigAudioBufferInfo_v5( xmlconfigstruct )
%
% This searches an XML configuration parse tree for an "Audio" tag,
% and extracts audio buffer information from it.
%
% "xmlconfigstruct" is a structure containing an XML parse tree of an
%   Open Ephys v0.5 config file, as returned by readstruct().
%
% "audiometa" is a structure with the following fields:
%   "samprate" is the audio sampling rate.
%   "bufsamps" is the length of the audio buffer in samples.
%   "bufms" is the length of the audio buffer in milliseconds. This is
%     the polling rate for Open Ephys signal chain updates.
```

13.10 nlOpenE_parseConfigProcessorsXML_v5.m

```
% function metalist = nlOpenE_parseConfigProcessorsXML_v5( xmlconfigstruct )
%
% This searches an XML configuration parse tree for processor nodes, and
% attempts to extract configuration metadata for each of them.
%
% "xmlconfigustruct" is a structure containing an XML parse tree of an
%   Open Ephys v0.5 config file, as returned by readstruct().
%
% "metalist" is a cell array with one entry per processor node found. Each
%   entry is a processor node metadata structure containing configuration
%   information in the format described in PROCMETA_OPENEPHYSv5.txt.
```

13.11 nlOpenE_parseProcessorNodeXML_v5.m

```
% function procmeta = nlOpenE_parseProcessorNodeXML_v5( xmlstruct )
%
% This parses an Open Ephys v0.5 XML processor node configuration tag and
% extracts configuration metadata from it.
%
```

```
% This is an "entry point" function; it extracts information common to all
% nodes and then calls helper functions to extract additional information
% specific to individual plugins and node types.
%
% "xmlstruct" is a structure containing an XML parse tree (per readstruct()).
%   This should be the parse tree for a "processor" node from an Open Ephys
%   v0.5 configuration file.
%
% "procmeta" is a structure containing node metadata, or struct([]) if an
%   error occurred. Node metadata is described in PROCMETA_OPENEPHYSv5.txt.
```

13.12 nlOpenE_parseProcessorXMLv5_ACCConditionalTrig.m

```
% function newmeta = nlOpenE_parseProcessorXMLv5_ACCConditionalTrig( ...
%   oldmeta, xmlproc, xmleditor )
%
% This parses an Open Ephys v0.5 XML processor node configuration tag for
% an ACC Lab Conditional Trigger plugin instance, and adds type-specific
% metadata to the supplied metadata structure.
%
% "oldmeta" is the metadata structure built by the "entry point" function.
%   It contains the "common" metadata described in PROCMETA_OPENEPHYSv5.txt.
% "xmlproc" is a structure containing the XML parse tree (per readstruct())
%   of the "processor" tag being interpreted.
% "xmleditor" is a structure containing the XML parse tree of the "editor"
%   tag within the "processor" tag, or struct([]) if none was found.
%
% "newmeta" is a copy of "oldmeta" augmented with plugin-specific metadata.
```

13.13 nlOpenE_parseProcessorXMLv5_ArduinoOut.m

```
% function newmeta = nlOpenE_parseProcessorXMLv5_ArduinoOut( ...
%   oldmeta, xmlproc, xmleditor )
%
% This parses an Open Ephys v0.5 XML processor node configuration tag for
% an Arduino Output node, and adds type-specific metadata to the supplied
% metadata structure.
%
% "oldmeta" is the metadata structure built by the "entry point" function.
%   It contains the "common" metadata described in PROCMETA_OPENEPHYSv5.txt.
% "xmlproc" is a structure containing the XML parse tree (per readstruct())
%   of the "processor" tag being interpreted.
% "xmleditor" is a structure containing the XML parse tree of the "editor"
%   tag within the "processor" tag, or struct([]) if none was found.
%
% "newmeta" is a copy of "oldmeta" augmented with plugin-specific metadata.
```

13.14 nlOpenE_parseProcessorXMLv5_Bandpass.m

```
% function newmeta = nlOpenE_parseProcessorXMLv5_Bandpass( ...
%   oldmeta, xmlproc, xmleditor )
%
% This parses an Open Ephys v0.5 XML processor node configuration tag for
% a bandpass filter node, and adds type-specific metadata to the supplied
% metadata structure.
%
% "oldmeta" is the metadata structure built by the "entry point" function.
%   It contains the "common" metadata described in PROCMETA_OPENEPHYSv5.txt.
% "xmlproc" is a structure containing the XML parse tree (per readstruct())
%   of the "processor" tag being interpreted.
% "xmleditor" is a structure containing the XML parse tree of the "editor"
%   tag within the "processor" tag, or struct([]) if none was found.
%
% "newmeta" is a copy of "oldmeta" augmented with plugin-specific metadata.
```

13.15 nlOpenE_parseProcessorXMLv5_ChannelMap.m

```
% function newmeta = nlOpenE_parseProcessorXMLv5_ChannelMap( ...
%   oldmeta, xmlproc, xmleditor )
%
% This parses an Open Ephys v0.5 XML processor node configuration tag for
% a channel mapping node, and adds type-specific metadata to the supplied
% metadata structure.
%
% "oldmeta" is the metadata structure built by the "entry point" function.
%   It contains the "common" metadata described in PROCMETA_OPENEPHYSv5.txt.
% "xmlproc" is a structure containing the XML parse tree (per readstruct())
%   of the "processor" tag being interpreted.
% "xmleditor" is a structure containing the XML parse tree of the "editor"
%   tag within the "processor" tag, or struct([]) if none was found.
%
% "newmeta" is a copy of "oldmeta" augmented with plugin-specific metadata.
```

13.16 nlOpenE_parseProcessorXMLv5_FileReader.m

```
% function newmeta = nlOpenE_parseProcessorXMLv5_FileReader( ...
%   oldmeta, xmlproc, xmleditor )
%
% This parses an Open Ephys v0.5 XML processor node configuration tag for
% a file reader node, and adds type-specific metadata to the supplied
% metadata structure.
%
```

```
% "oldmeta" is the metadata structure built by the "entry point" function.
%   It contains the "common" metadata described in PROCMETA_OPENEPHYSv5.txt.
% "xmlproc" is a structure containing the XML parse tree (per readstruct())
%   of the "processor" tag being interpreted.
% "xmleditor" is a structure containing the XML parse tree of the "editor"
%   tag within the "processor" tag, or struct([]) if none was found.
%
% "newmeta" is a copy of "oldmeta" augmented with plugin-specific metadata.
```

13.17 nlOpenE_parseProcessorXMLv5_FileRecord.m

```
% function newmeta = nlOpenE_parseProcessorXMLv5_FileRecord( ...
%   oldmeta, xmlproc, xmleditor )
%
% This parses an Open Ephys v0.5 XML processor node configuration tag for
% a file-writing node ("Record Node"), and adds type-specific metadata to
% the supplied metadata structure.
%
% "oldmeta" is the metadata structure built by the "entry point" function.
%   It contains the "common" metadata described in PROCMETA_OPENEPHYSv5.txt.
% "xmlproc" is a structure containing the XML parse tree (per readstruct())
%   of the "processor" tag being interpreted.
% "xmleditor" is a structure containing the XML parse tree of the "editor"
%   tag within the "processor" tag, or struct([]) if none was found.
%
% "newmeta" is a copy of "oldmeta" augmented with plugin-specific metadata.
```

13.18 nlOpenE_parseProcessorXMLv5_IntanRec.m

```
% function newmeta = nlOpenE_parseProcessorXMLv5_IntanRec( ...
%   oldmeta, xmlproc, xmleditor )
%
% This parses an Open Ephys v0.5 XML processor node configuration tag for
% an Intan recording controller node, and adds type-specific metadata to
% the supplied metadata structure.
%
% "oldmeta" is the metadata structure built by the "entry point" function.
%   It contains the "common" metadata described in PROCMETA_OPENEPHYSv5.txt.
% "xmlproc" is a structure containing the XML parse tree (per readstruct())
%   of the "processor" tag being interpreted.
% "xmleditor" is a structure containing the XML parse tree of the "editor"
%   tag within the "processor" tag, or struct([]) if none was found.
%
% "newmeta" is a copy of "oldmeta" augmented with plugin-specific metadata.
```

13.19 nlOpenE_parseProcessorXMLv5_TNECrossingDetector.m

```
% function newmeta =nlOpenE_parseProcessorXMLv5_TNECrossingDetector( ...
%   oldmeta, xmlproc, xmleditor )
%
% This parses an Open Ephys v0.5 XML processor node configuration tag for
% a TNE Lab Crossing Detector plugin instance, and adds type-specific
% metadata to the supplied metadata structure.
%
% "oldmeta" is the metadata structure built by the "entry point" function.
%   It contains the "common" metadata described in PROCMETA_OPENEPHYSv5.txt.
% "xmlproc" is a structure containing the XML parse tree (per readstruct())
%   of the "processor" tag being interpreted.
% "xmleditor" is a structure containing the XML parse tree of the "editor"
%   tag within the "processor" tag, or struct([]) if none was found.
%
% "newmeta" is a copy of "oldmeta" augmented with plugin-specific metadata.
```

13.20 nlOpenE_parseProcessorXMLv5_TNEPhaseCalculator.m

```
% function newmeta = nlOpenE_parseProcessorXMLv5_TNEPhaseCalculator( ...
%   oldmeta, xmlproc, xmleditor )
%
% This parses an Open Ephys v0.5 XML processor node configuration tag for
% a TNE Lab Phase Calculator plugin instance, and adds type-specific
% metadata to the supplied metadata structure.
%
% "oldmeta" is the metadata structure built by the "entry point" function.
%   It contains the "common" metadata described in PROCMETA_OPENEPHYSv5.txt.
% "xmlproc" is a structure containing the XML parse tree (per readstruct())
%   of the "processor" tag being interpreted.
% "xmleditor" is a structure containing the XML parse tree of the "editor"
%   tag within the "processor" tag, or struct([]) if none was found.
%
% "newmeta" is a copy of "oldmeta" augmented with plugin-specific metadata.
```

13.21 nlOpenE_probeFolder.m

```
% function foldermeta = nlOpenE_probeFolder( indir )
%
% This checks for the existence of Open Ephys format data files in the
% specified folder, and constructs a folder metadata structure if data is
% found.
%
% If no data is found, an empty structure is returned.
```

```
%
% "indir" is the directory to search.
%
% "foldermeta" is a folder metadata structure, per FOLDERMETA.txt.
```

Part III

Application Libraries

Chapter 14

Application Library Structures and Additional Notes

14.1 CHANREC.txt

```
A "channel record" is a structure with the following fields:

"folder" is the folder ID of the channel.
"bank" is the bank ID of the channel.
"chan" is the channel number of the channel.
"result" is the processing result associated with this channel.

The intention is that a vector of channel records may be returned that
describes a ranked list of channels along with statistical analysis results
associated with each channel.
```

14.2 TUNING.txt

```
Tuning parameter structures are used by several processing functions. These
structures are defined as follows.


An "artifact rejection tuning parameter" structure is used by
nlChan_applyArtifactReject(), and a default version is provided by
nlChan(getArtifactDefaults(). It has the following fields:

- "trimstart" is the number of seconds to remove at the start of the signal.
- "trimend" is the number of seconds to remove at the end of the signal.
- "ampthresh" is the high threshold for amplitude-based artifact detection.
- "amphalo" is the low threshold for amplitude-based artifact detection.
- "diffthresh" is the high threshold for derivative-based artifact detection.
```

- "diffhalo" is the low threshold for derivative-based artifact detection.
- "timehalosecs" is the additional time in seconds to squash around artifacts.
- "smoothsecs" is the time window size to use for smoothing the derivative.
- "dcsecs" is the time window size to use for DC removal.

See documentation for nlProc_removeArtifactsSigma() for further details of
the artifact rejection algorithm.

A "filter tuning parameter" structure is used by nlChan_applyFiltering(), and
a default version is provided by nlChan_getFilterDefaults(). It has the
following fields:

- "lfprate" is the sampling rate to use when generating the low-pass-filtered
  signal (LFP signal).
- "lfpcorner" is the sampling rate to use when splitting the signal into
  a low-pass-filtered signal (LFP signal) and high-pass-filtered signal
  (spike signal).
- "powerfreq" is a scalar or vector containing power line notch filter values
  to apply. This is typically the power line frequency and its harmonics.
- "dcfreq" is the corner frequency to use for the DC removal filter.

See documentation for nlProc_filterSignal() for further details of signal
filtering.

A "percentile tuning parameter" structure is used by nlChan_processChannel()
and by the "Channel Analysis Tool". It has the following fields:

- "burstrange" is a vector containing percentile values to use when looking
  for burst activity. This should be sorted in ascending order.
- "burstselectidx" is the index of the entry in "burstrange" to default to.
- "spikerange" is a vector containing percentile values to use when looking
  for spike activity. This should be sorted in ascending order.
- "spikeselectidx" is the index of the entry in "spikerange" to default to.

See documentation for nlProc_calcSkewPercentile() and documentation for
nlProc_calcSpectrumSkew() for further details of outlier identification for
activity detection.

A "spectrum tuning parameter" structure is used by nlChan_processChannel().
It has the following fields:

- "freqlow" is the minimum frequency to tabulate information for.
- "freqhigh" is the maximum frequency to tabulate information for.
- "freqsperdecade" specifies the spacing of frequency bins.

- "winsecs" is the time window size to use, in seconds.
- "winsteps" is the number of overlapping steps taken when advancing the
  time window. It advances "winsecs/winsteps" seconds per step.

See documentation for nlProc_calcSpectrumSkew() for further details of
outlier identification in time-frequency spectrograms.

This is the end of the file.

Chapter 15

"nlChan" Functions

15.1 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).
```

15.2 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.
```

15.3 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.
```

15.4 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".
```

15.5 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".
```

15.6 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".
```

15.7 nlChan_getSpectrumDefaults.m

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

15.8 nlChan_processChannel.m

```
% function resultstats = nlChan_processChannel( wavedata, samprate, ...
%   refdata, tuningart, 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.
% "refdata" is the reference waveform. This should already be truncated and
%   have artifacts removed, but should retain NaN values to avoid introducing
%   new artifacts. If refdata is [], no re-referencing is performed.
% "tuningart" is a structure containing tuning parameters for artifact removal.
% "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.
```

15.9 nlChan_rankChannels.m

```
% function [ bestlist typbest typmiddle typworst ] = ...
%   nlChan_rankChannels( chanresults, maxperbank, typfrac, scorefunc )
%
% This process evaluates a result list returned by nlIO_iterateChannels().
% Channels receive a score, and a list of channel records is compiled that is
% sorted by that score. Channel records for "typical" best, worst, and
% middle-scoring entries are selected, and these plus a trimmed sorted list
% are returned.
%
% Channel records have the format given in "CHANREC.txt".
%
% The sorted list is trimmed to include at most a certain number of channels
% per bank.
%
% NOTE - The "typical" records are not necessarily in the trimmed result list.
%
% "chanresults" is a channel list per "CHANLIST.txt" that has been augmented
%   with "resultlist" fields by per-channel signal processing.
% "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 a channel's "resultlist" value, per
%     nlIO_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.
```

Chapter 16

"nlCheck" Functions

16.1 nlCheck_getFTSignalBits.m

```
% function chanbits = nlCheck_getFTSignalBits( ftdata )
%
% This computes the number of bits of dynamic range in each channel and trial
% in a Field Trip dataset.
%
% The number will only be meaningful with integer data (i.e. with a minimum
% step size of 1.0).
%
% This will usually be called with single-trial continuous data to provide a
% sanity check of recording settings.
%
% "ftdata" is a ft_datatype_raw structure containing ephys data.
%
% "chanbits" is a cell array with one cell per trial, each containing a
%   Nchans x 1 floating-point vector with the number of bits needed to
%   represent each channel's data in that trial.
```

16.2 nlCheck_getSignalBits.m

```
% function signalbits = nlCheck_getSignalBits( wavedata )
%
% This computes the number of bits of dynamic range in a signal.
% The number will only be meaningful with integer data (i.e. with a minimum
% step size of 1.0).
%
% "wavedata" is a vector containing waveform data samples.
%
% "signalbits" is a floating-point value containing the number of bits
%   needed to represent the waveform data.
```

16.3 nlCheck_testDropoutsArtifacts.m

```
% function [ dropoutfrac artifactfrac ] = nlCheck_testDropoutsArtifacts( ...
%   wavedata, smoothsamps, dropout_threshold, artifact_threshold )
%
% This function computes a rectified version of the input signal, smooths it,
% finds the median amplitude, and looks for samples that are above some
% multiple of the median amplitude or below some fraction of the median
% amplitude.
%
% Samples above the high threshold are assumed to be artifacts, and samples
% below the low threshold are assumed to be drop-outs.
%
% "wavedata" is a vector containing waveform data samples.
% "smoothsamps" is the smoothing window size, in samples. This is
%   approximately one period at the low-pass filter's cutoff frequency.
% "dropout_threshold" is a multiplier determining the lower threshold to test.
%   This should be less than one.
% "artifact_threshold" is a multiplier determining the upper threshold to
%   test. This should be greater than one.
%
% "dropoutfrac" is the fraction of samples that were below the lower
%   threshold.
% "artifactfrac" is the fraction of samples that were above the upper
%   threshold.
```

16.4 nlCheck_testFTDropoutsArtifacts.m

```
% function [ dropoutfrac artifactfrac ] = nlCheck_testFTDropoutsArtifacts( ...
%   ftdata, smoothfreq, dropout_threshold, artifact_threshold )
%
% This function calls nlCheck_testDropoutsArtifacts() to compute the fraction
% of samples that are dropouts and the fraction of samples that are artifacts
% in each channel and trial in a Field Trip dataset.
%
% This is done by rectifying and smoothing the original signal and then
% thresholding the resulting signal against multiples of the median signal.
% This should be treated as an estimate only; for more accurate artifact and
% dropout detection, use more sophisticated algorithms.
%
% "ftdata" is a ft_datatype_raw structure containing ephys data.
% "smoothfreq" is the low-pass filter corner frequency to use for smoothing
%   the data prior to detection. Artifacts and dropouts shorter than 1/2pi*f
%   will be attenuated.
% "dropout_threshold" is a multiplier determining the lower threshold to test.
%   This should be less than one.
% "artifact_threshold" is a multiplier determining the upper threshold to
```

```
%   test. This should be greater than one.
%
% "dropoutfrac" is a cell array with one cell per trial, each containing a
%   Nchans x 1 floating-point vector with fraction of samples that were below
%   the lower threshold.
% "artifactfrac" is a cell array with one cell per trial, each containing a
%   Nchans x 1 floating-point vector with fraction of samples that were above
%   the upper threshold.
```

Part IV

Sample Code

Chapter 17

Sample Code

17.1 _.m