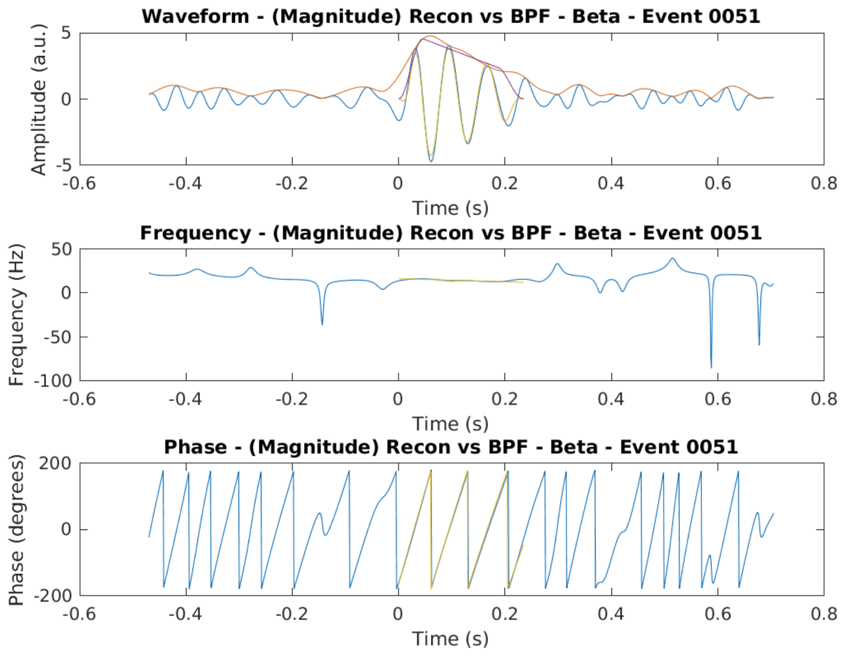
wlBurst Library – User Guide

Written by Christopher Thomas – September 15, 2020.



Contents

Chapter 1

Overview

1.1 Introduction

Neural signals often contain transient oscillations in the local field potential ("oscillatory bursts"). These represent coordinated activity of many neurons. There is evidence that oscillatory bursts are correlated with changes in connectivity within the brain and with certain types of data processing, so identifying and characterizing these bursts in neural recordings is useful.

An illustration of burst events within a neural signal is shown in Figure 1.1.

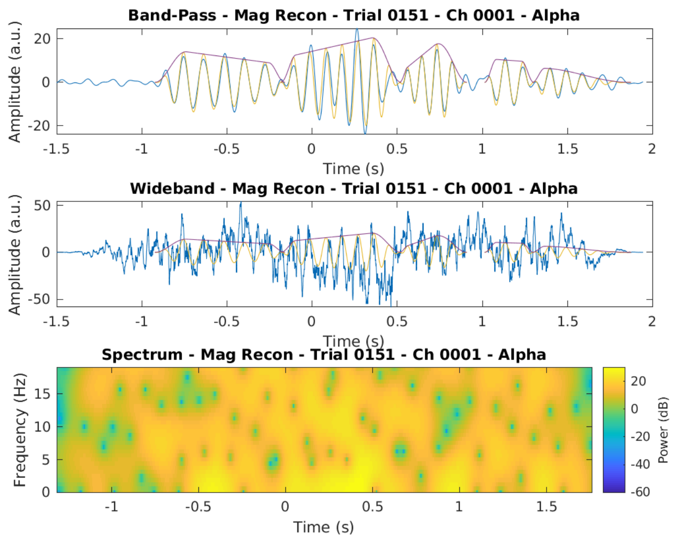


Figure 1.1: Neural waveform with burst events.

The purpose of the wlBurst library is to provide the tools needed to identify and characterize oscillatory bursts.

1.2 The wlBurst Library

The wlBurst library primarily uses the following types of data structure:

- **"Data traces"** are one-dimensional arrays containing waveform samples.
- **"Event lists"** are one-dimensional arrays containing "event records" (per `EVENTFORMAT.txt`). These are parametric descriptions of oscillation events extracted from a data trace.
- **"Field Trip data structures"** are structures of type `ft_datatype_raw` produced by the Field Trip toolset. The wlBurst library treats trial data within a FT data structure as a two-dimensional array of data traces (indexed by trial number and channel number).
- **"Event matrix"** structures are three-dimensional arrays of event lists, describing events detected within a Field Trip data structure indexed by frequency band, trial number, and channel number. A considerable amount of auxiliary data is stored as well (per `EVMATRIX.txt`).

  Event matrices may be converted to Field Trip data structures, to allow the use of the Field Trip suite's visualization tools. This stores events as individual FT trials. Information mapping events to the original band, trial, and channel numbers is stored in the `trialinfo` structure, and additional metadata is stored in a "`wlnotes`" structure per `WLNOTES.txt`.

Library functions are divided into the following categories:

- **"Processing"** functions (in `lib-wl-proc`) perform segmentation, feature extraction, and other operations on single data traces. Events are returned as "event lists", which can be further manipulated.
- **"Field Trip"** functions (in `lib-wl-ft`) perform event detection and other operations on Field Trip raw data structures. Events detected are returned as "event matrices", which can be further manipulated.
- **"Synthesis"** functions (in `lib-wl-synth`) construct simulated neural recordings containing oscillation events, providing data with known ground-truth.
- **"Auxiliary"** functions (in `lib-wl-aux`) perform manipulations that don't fall into the previous categories.
- **"Plotting"** functions (in `lib-wl-plot`) produce rough plots of various types of data. The output is generally not publication-quality.

Library functions are documented in detail in the "Function Reference" document.

Chapter 2

Segmentation and Parametric Representation

2.1 Segmentation – Detecting Burst Events

"Segmentation" is the process of deciding which portions of an input signal contain oscillatory bursts. Historically, this has usually been done by performing a Hilbert transform to get an analytic signal with known magnitude and phase, and looking for magnitude excursions above some threshold (usually 3σ with respect to the mean magnitude across all parts of the waveform and all trials). Many variants of this approach exist, and several other approaches have been used, but these are beyond the scope of this document.

The wlBurst library provides an implementation of magnitude-threshold detection, using the algorithm illustrated in Figure 2.1. Rather than using a fixed threshold, this implementation uses a low-pass-filtered version of the magnitude as its "DC" level, and looks for magnitude excursions with respect to that (comparing to a local rather than global mean). The intention is that this is less sensitive to drift in signal properties.

A typical event segmented by magnitude thresholding is shown in Figure 2.2. Notice that the analytic magnitude (orange envelope in the top pane) is only well-behaved for the band-pass-filtered signal.

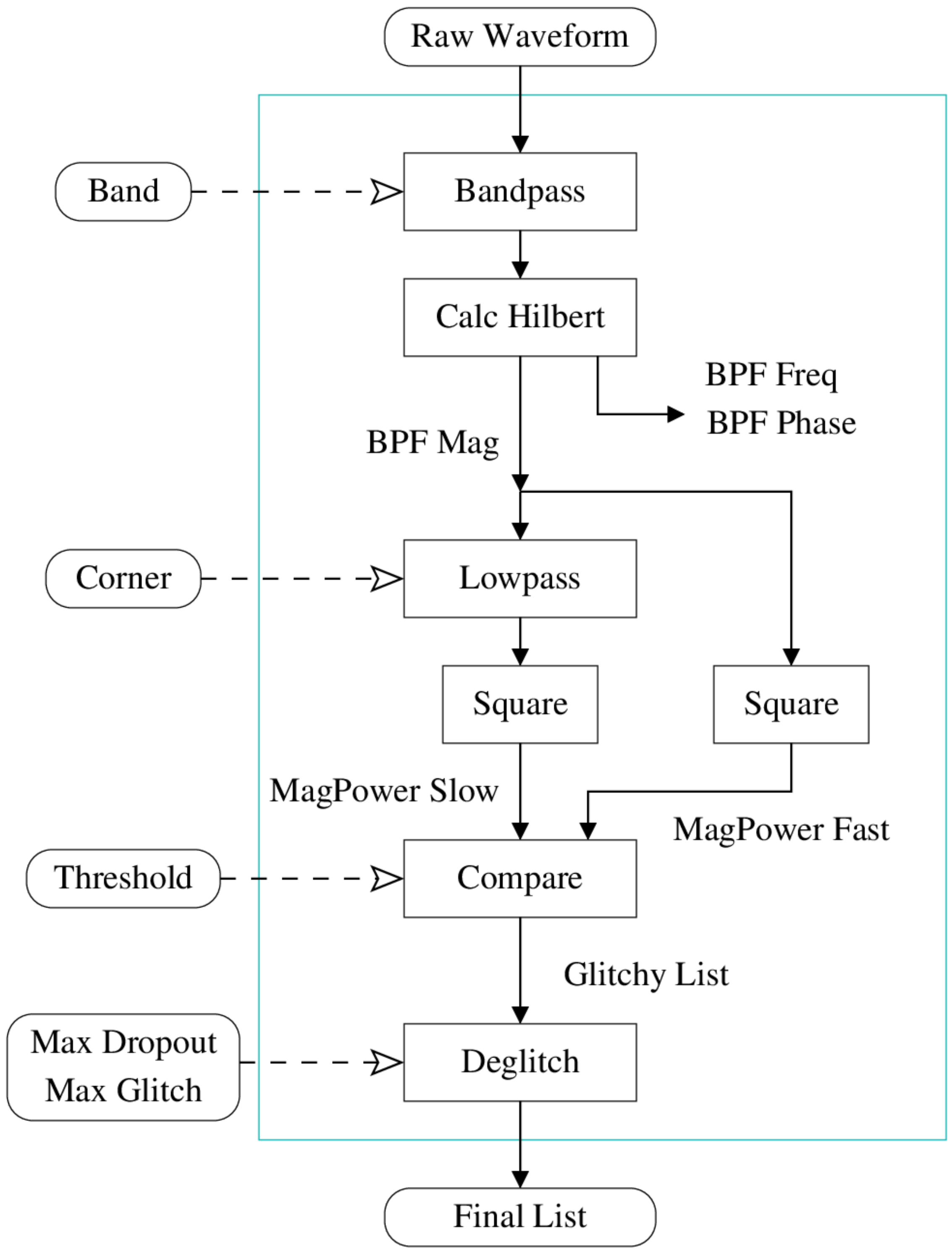
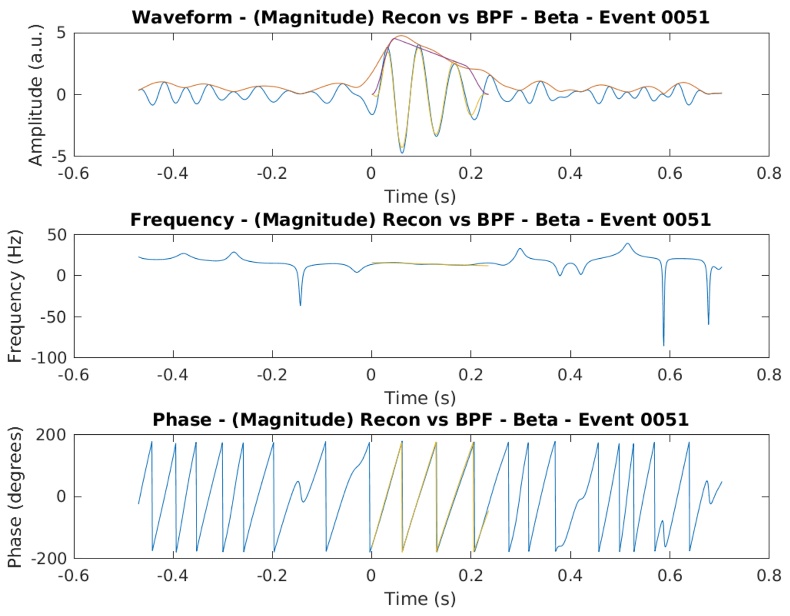
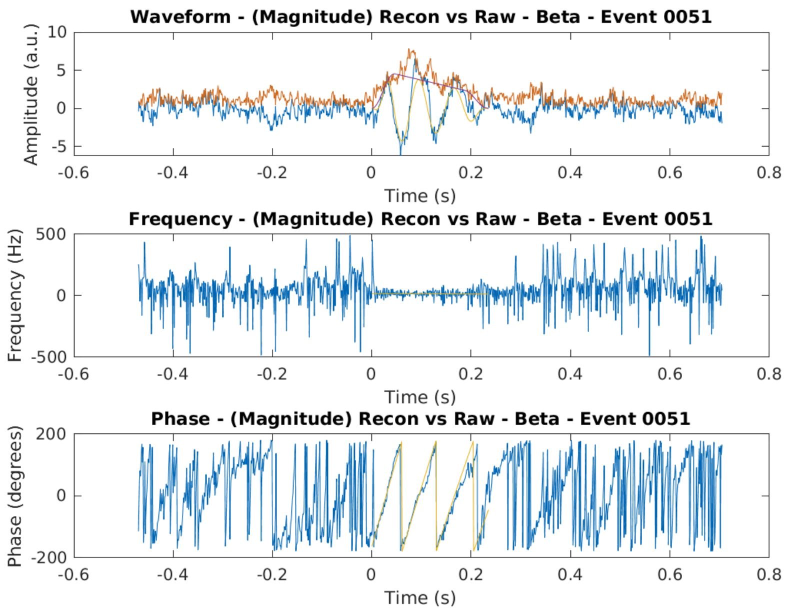


Figure 2.1: Segmentation by Magnitude Thresholding



Bandpass



Wideband

Figure 2.2: Typical event segmented by magnitude thresholding.

In addition to magnitude-threshold event detection, the wlBurst library provides an implementation of “frequency-stability” event detection. This algorithm looks at the frequency of the analytic signal (middle panes in Figure 2.2. The analytic frequency is defined as the derivative of the phase of the analytic signal (which is provided directly by the Hilbert transform).

During oscillation events, the frequency is expected to be well-defined (staying within a narrow range with few or no excursions). Outside of oscillation events, when the neural signal is not oscillating with a well-defined frequency, excursions in analytic frequency are expected to occur. These two situations are difficult to distinguish in the Hilbert transform of the band-pass-filtered waveform, but are visually apparent in the Hilbert transform of the wideband signal.

To perform automated segmentation, high-frequency noise with known properties is added to the band-pass-filtered signal. The Hilbert transform of this “noisy” signal is computed, and the variance of the resulting analytic frequency signal is measured. This algorithm is illustrated in Figure 2.3.

A typical event segmented by frequency stability detection is shown in Figure 2.4. Notice that the wideband signal shows out-of-band events, and that the band-pass-filtered signal with added noise shows only in-band events and has more consistenly different variance in the frequency signal between event and non-event regions. The “noisy” signal is only used for detection - parameter extraction uses the clean band-pass-filtered signal.

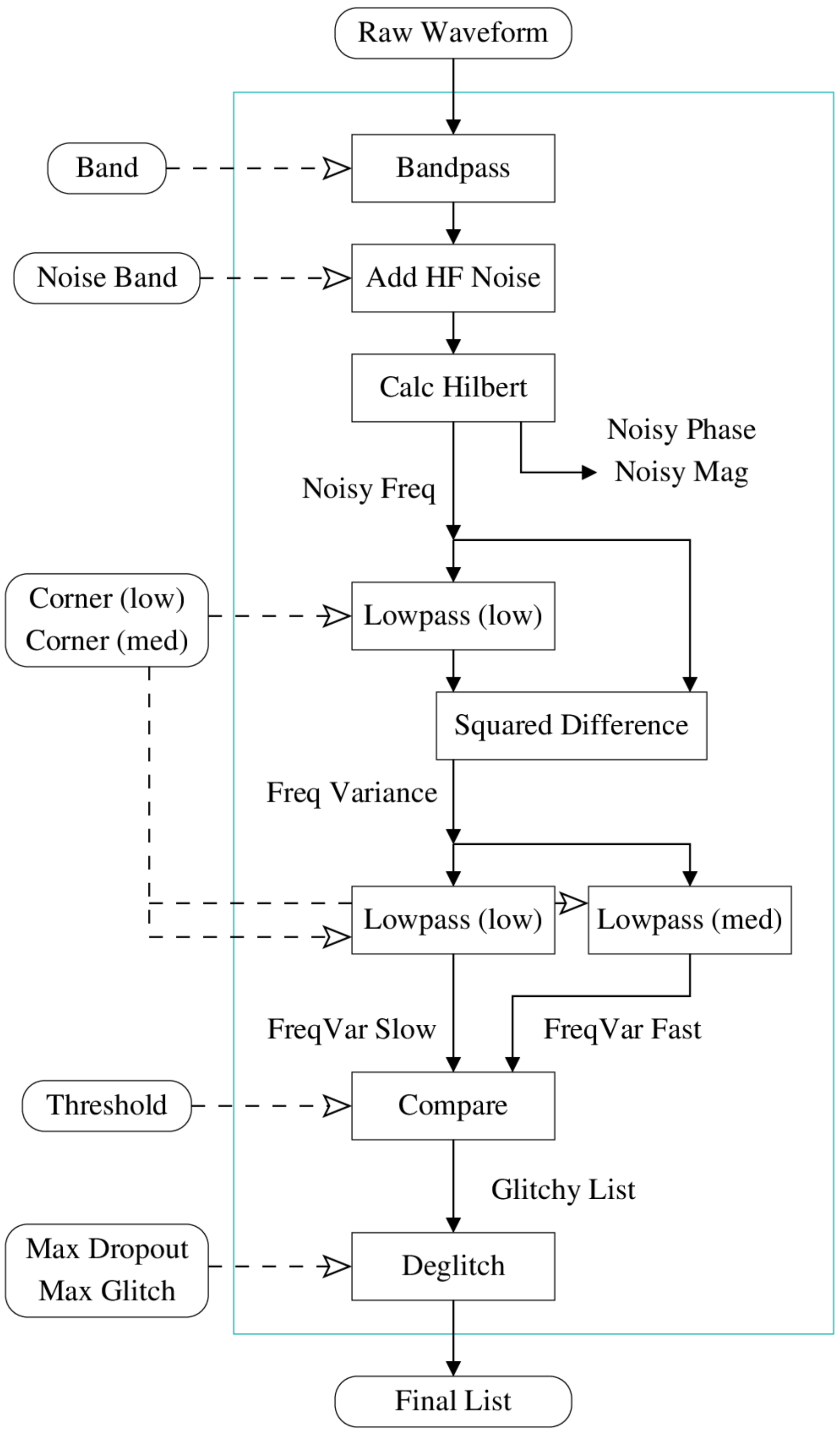
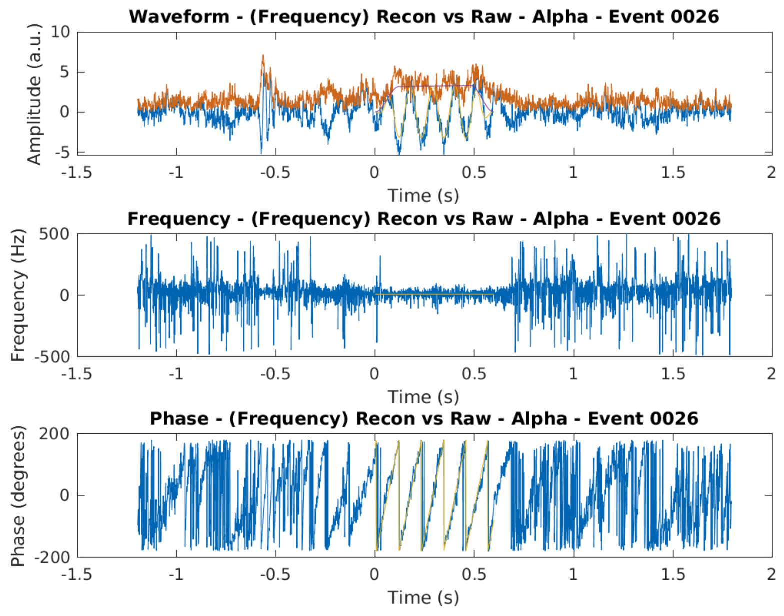
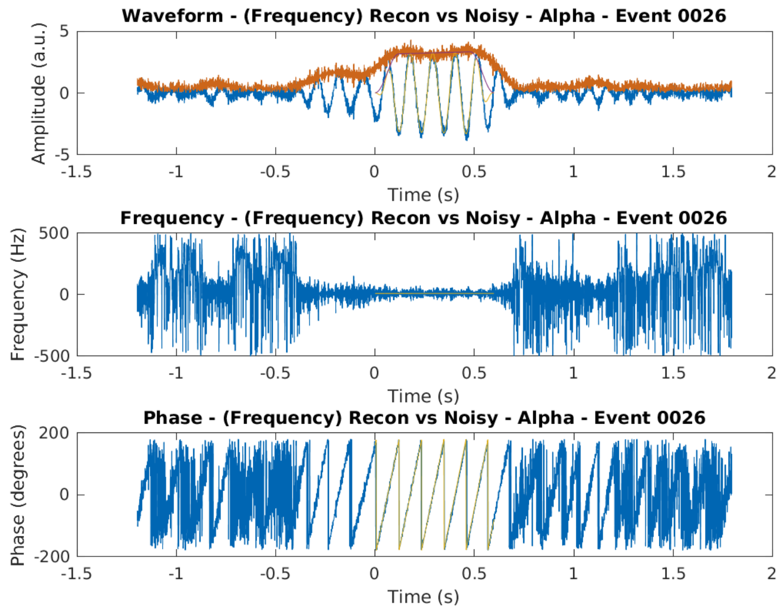


Figure 2.3: Segmentation by Frequency Stability



Wideband



BPF Plus Noise

Figure 2.4: Typical event segmented by frequency stability.

2.2 Feature Extraction – Curve-Fitting Oscillatory Bursts

"Feature extraction" or "parameter extraction" is the process of building a description of a burst event that contains all desired information about the burst. This is usually done as a separate step after segmentation (burst detection). The model used to represent oscillatory bursts in the wlBurst library is shown in Figure 2.5, and is described in `EVENTFORMAT.txt` (as the "chirpramp" model). Future versions of the wlBurst library will explicitly support additional models; for the time being, custom models may be used provided care is taken to not call analysis or plotting functions that assume use of the "chirpramp" model.

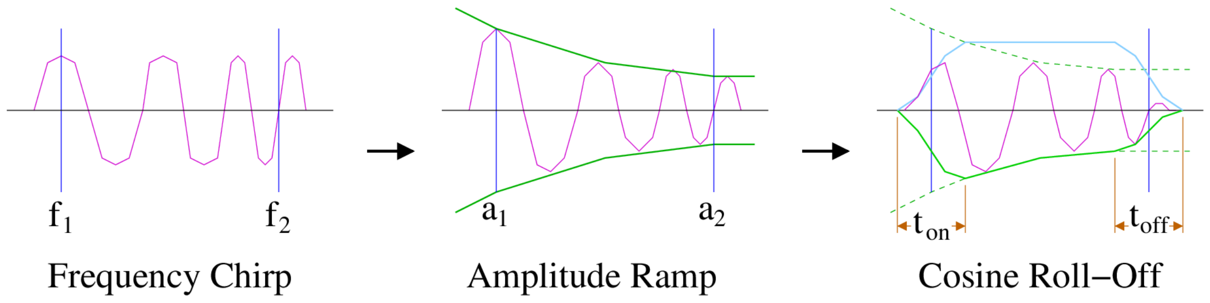


Figure 2.5: Parametric burst model ("`chirpramp`" model).

"Chirpramp" model fits are performed through several steps:

- Endpoints of the burst are assumed. These may either be the times when the detection criterion (absolute magnitude or frequency variance) crossed its threshold, or where the detection criterion crosses some other lower threshold (for "dual" variants of the magnitude-threshold and frequency-stability detection algorithms).
- The magnitude envelope is curve-fit. This involves choosing roll-on and roll-off times, and performing exponential and linear fits to the analytic magnitude and choosing the best match. Choosing roll-on and roll-off times may be done via a grid search (fast but less accurate) or by simulated annealing (slow but more accurate).
- The frequency and phase are curve-fit. This involves performing exponential and linear fits to the analytic frequency and choosing the best match, followed by testing possible starting phases and choosing the one that best matches modeled burst phase and the analytic phase across the duration of the burst.
- Optionally, a simulated annealing algorithm may perturb all components of the parametric burst to try to better match the input waveform. This is slow but slightly improves accuracy.

User-specified segmentation and parameter extraction algorithms may alternatively be used, per `SEGCONFIG.txt` and `PARAMCONFIG.txt`.

Chapter 3

Tutorial

3.1 Reading and Processing Field Trip Data

A minimal script for reading and processing Field Trip data using the wlBurst library is given in Section 3.2. Steps that it performs are described below:

- Libraries are added to the path.
- Frequency bands of interest are defined.

  The structure format used for this follows the same conventions as the "`bandinfo`" structure array in `WLNOTES.txt`.
- Segmentation and parameter extraction configuration structures are created. These use the conventions described in `SEGCONFIG.txt` and `PARAMCONFIG.txt`, respectively.
- Per-band overrides for segmentation and parameter extraction are defined, per `wlFT_doFindEventsInTrials_MT.m`.

  These are intended to provide band-specific values for some of the fields in the segmentation and parameter extraction configuration structures. The sample code overrides several segmentation configuration fields but none of the parameter extraction fields.

  Tuning detection threshold ("`dbpeak`") is particularly important. A threshold that reliably detects events in one band may produce too many false positives or fail to detect most events in another band.
- The Field Trip data is loaded (in `ft_datatype_raw` format), and is pruned to the desired subset of channels (the single channel containing local field potential recordings, for the example dataset).
- Metadata from a custom structure recording reward times is loaded and converted into a data column in `trialinfo` within the Field Trip data. This is then used to trim trials to exclude regions known to have electrical artifacts.

  The type of trimming that will be appropriate will vary widely from use case to use case, but some type of trimming and/or time alignment will usually be needed.

- A 60 Hz notch filter is applied (FWHM 6 Hz, 20th order). This is intended to remove electrical artifacts due to power line noise at US power frequencies that existed in this dataset.

  The use of `filtfilt` means this filter is applied twice (forwards and backwards in time to avoid any net phase offset). The resulting operation is equivalent to a 40th-order filter with zero phase shift.

- Events are detected, via a call to `wlFT_doFindEventsInTrials_MT`, passing the configuration structures that were previously defined.

  Custom detection algorithms may be implemented using the "`custom`" type described in `SEGCONFIG.txt` and `PARAMCONFIG.txt`.

- Events near the start and end of the traces are trimmed, to reduce spurious event detections caused by the roll-off window applied at the ends of the trace during detection.

- Detected events are plotted against their trial waveforms.

  Several hundred trials are present, so the "trial stride" argument is used to plot only the first trial out of every 25, for demonstration purposes.

- Detected events are saved.

  Event detection is typically time-consuming, especially if annealing is used to determine event parameters. Saving and reloading the detected event list is usually better than re-detecting events when performing post-processing of event data.

Typical plots generated by this script are shown in Figures 3.1 and 3.2.

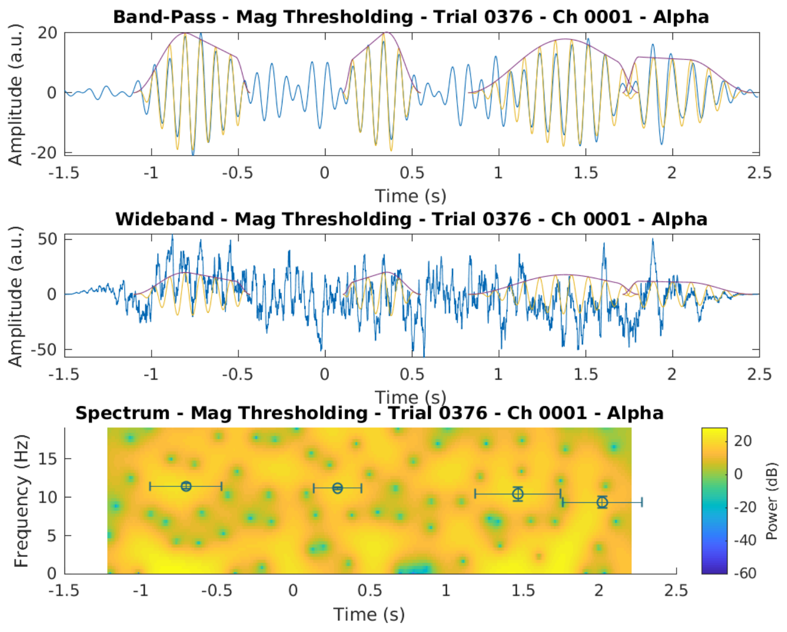


Figure 3.1: Field Trip data - typical detected events (alpha).

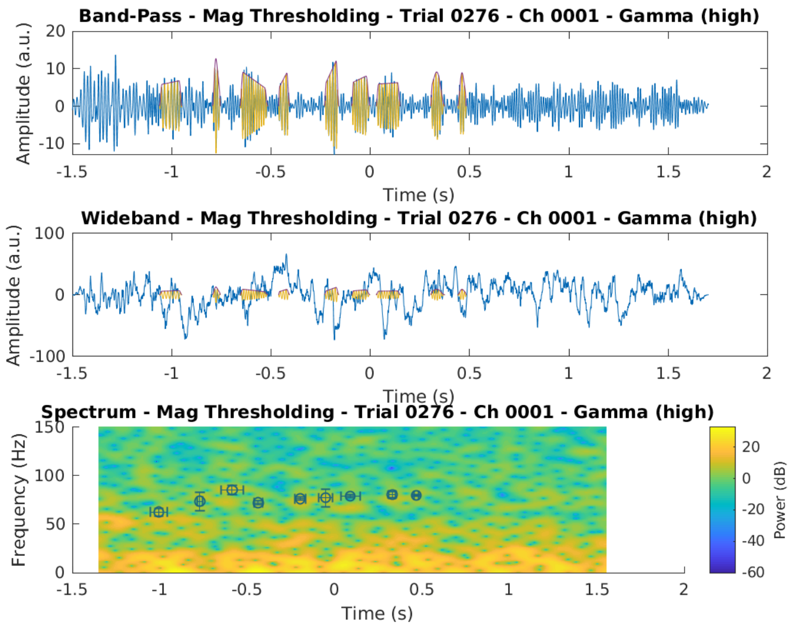


Figure 3.2: Field Trip data - typical detected events (gamma).

3.2 Minimal Field Trip Processing Script

3.2.1 do_minimal_ft.m

```
% Minimal tutorial script - Extracting burst events from FT data.
% See LICENSE.md for distribution information.

%
%
% Configuration.

%
% Paths.

addpath('lib-wl-synth', 'lib-wl-proc', 'lib-wl-ft', 'lib-wl-aux', ...
  'lib-wl-plot');


%
% Data files.

inputfile = 'thilo-data/LFP_B_tmplate_01.mat';
outputfile = 'output/minimal-ftevents.mat';
plotdir = 'output';


%
% Bands of interest.

% NOTE - Splitting gamma, as detection works best if bands are about one
% octave wide.
bandlist = [ ...
  struct('band', [ 4 7.5 ],    'label', 'th', 'name', 'Theta') ...
  struct('band', [ 7.5 12.5 ], 'label', 'al', 'name', 'Alpha') ...
  struct('band', [ 12.5, 30 ], 'label', 'be', 'name', 'Beta') ...
  struct('band', [ 30 60 ],    'label', 'gl', 'name', 'Gamma (low)') ...
  struct('band', [ 60 100 ],   'label', 'gh', 'name', 'Gamma (high)') ];


%
% Segmentation and parameter extraction.

% NOTE - The "3 sigma" rule of thumb is 9.5 dB.
% For this data, we need to be a bit more sensitive than that.


% Detection by magnitude thresholding.

segconfig = struct( 'type', 'magdual', ...
  'qlong', 10, 'qdrop', 1.0, 'qglitch', 1.0, ...
  'dbpeak', 6, 'dbend', 2 );

% Envelope fitting by a coarse grid search.

paramconfig = struct( 'type', 'grid', 'gridsteps', 7 );


% Band-specific detection tuning.

seg_dbpeak = [ 6, 7, 6, 5, 4 ];
seg_dbend =  [ 5, 2, 1, 1, 0 ];
seg_qlong =  [ inf, 10, 10, 10, inf ];  % Use DC average for theta, gamma_hi
seg_qdrop =  [ 0.25, 0.5, 0.5, 1.0, 3.0 ];

for bidx = 1:length(bandlist)

  bandoverrides(bidx) = struct( 'seg', struct( ...
    'dbpeak', seg_dbpeak(bidx), 'dbend', seg_dbend(bidx), ...
    'qlong', seg_qlong(bidx), 'qdrop', seg_qdrop(bidx) ...
    ), 'param', struct() );

end



%
%
% Load the FT data.


load(inputfile, 'DATA');

% Get the sampling rate.
ftrate = wlFT_getSamplingRate(DATA);

% Extract our desired channels. Just one, for this dataset.
[ rawdata chandefs ] = wlFT_getChanSubset( DATA, { 'CSC5LFP' } );



%
%
% Preprocessing: Trim the trials.
% We have large electrical artifacts near reward time.


load(inputfile, 'tinfo');
```

```
% We're given a negative timestamp in seconds.
% Convert this into a positive sample offset.

thilo_rewardsamps = [];
scratch = tinfo.rewOnTimes;

for tidx = 1:length(scratch);
  thilo_rewardsamps(tidx) = round( - ftrate * scratch{tidx}(1) );
end


% Add the stop time as an extra field in "trialinfo".

auxidx = size(rawdata.trialinfo);
auxidx = auxidx(2) + 1;

rawdata.trialinfo(:,auxidx) = thilo_rewardsamps;


% Make a crop function that checks this new endpoint.

cropfunc = @(thistrial, sampinforow, trialinforow) ...
  deal( 1, trialinforow(auxidx) );

% Crop the trials.
[ trimmeddata cropdefs ] = wlFT_trimTrials(rawdata, cropfunc);



%
%
% Preprocessing: Apply a notch filter.


powerfilt = designfilt( 'bandstopiir', 'SampleRate', ftrate, ...
 'HalfPowerFrequency1', 57.0, 'HalfPowerFrequency2', 63.0, ...
 'FilterOrder', 20 );

for tidx = 1:length(trimmeddata.trial)

  thistrial = trimmeddata.trial{tidx};
  [ chancount, sampcount ] = size(thistrial);

  for cidx = 1:chancount
    thischannel = thistrial(cidx, :);
    thischannel = filtfilt(powerfilt, thischannel);
    thistrial(cidx, :) = thischannel;
  end

  trimmeddata.trial{tidx} = thistrial;
```

```
end



%
%
% Detect events in the FT data.


eventmatrix = wlFT_doFindEventsInTrials_MT( trimmeddata, bandlist, ...
  segconfig, paramconfig, bandoverrides, true );

% Clip events near the ends of the trial; roll-on and roll-off give
% spurious detections.

padtime = 0.4;
eventmatrix = wlFT_pruneEventsByTime(eventmatrix, padtime, padtime);

% Package events in Field Trip format, to use with FT's visualizers.
events_ft = wlFT_getEventTrialsFromMatrix(eventmatrix);



%
%
% Plot detected events.

figconfig = struct( ...
  'fig', figure, 'outdir', plotdir, ...
  'fsamp', ftrate, ...
  'psfres', 5, 'psolap', 99, 'psleak', 0.75, 'psylim', 50 );

plots_per_band = inf;
trial_stride = 25;
channel_stride = 1;

wlPlot_plotAllMatrixEvents(figconfig, eventmatrix, ...
  'Mag Thresholding', 'mag', plots_per_band, trial_stride, channel_stride);



%
%
% Save detected events.


% Align FT sample indices back to un-trimmed data.
events_ft = wlFT_unTrimMetadata(events_ft, cropdefs);
```

```
% Map FT channels back to the original channel indices.
events_ft = wlFT_unMapChannels(events_ft, chandefs);

save( outputfile, 'eventmatrix', 'cropdefs', 'chandefs', 'events_ft' );



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

3.3 Creating and Processing Synthetic Data

A minimal script for creating synthetic neural data, detecting burst events, and comparing detected events to ground truth is given in Section 3.4. Steps that it performs are described below:

- Libraries are added to the path.
- Frequency bands of interest are defined.

  The structure format used for this follows the same conventions as the "`bandinfo`" structure array in `WLNOTES.txt`.
- Segmentation and parameter extraction configuration structures are created. These use the conventions described in `SEGCONFIG.txt` and `PARAMCONFIG.txt`, respectively.
- Detection threshold levels for a brute-force threshold sweep are defined, as are per-band hand-tuned detection thresholds.
- Per-band overrides for segmentation and parameter extraction are defined, per `wlFT_doFindEventsInTrials_MT.m`.
- Synthetic neural data is generated via a call to `wlSynth_genFieldTrip`, using an array of parameter specification structures to define many different types of event. Ground truth for this synthetic data is also recorded.
- Event detection is performed via calls to `wlFT_doFindEventsInTrials_MT` (using magnitude-threshold detection), with the detection threshold swept across a range of values. Post-processing is applied (via `wlFT_calcEventErrors` and `wlAux_pruneMatrix`) to remove putative events with waveforms that don't match the band-pass-filtered input signal. The output of this step is a cell array of event matrices, indexed by threshold.
- A "selected" event matrix is constructed using per-band detected event lists corresponding to the per-band hand-tuned thresholds.
- False positive, false negative, and true positive detection rates are computed for each threshold in the threshold-sweep results, via `wlFT_compareMatrixEventsVsTruth`.
- Plots are generated for the following data:
  - Ground truth events in each trial are plotted against their trial waveforms.
  - Detected events with hand-tuned thresholds are plotted against their trial waveforms.
  - False positive and true positive rates (and counts) are plotted as a function of threshold.
  - Curve fit parameters for detected events with hand-tuned thresholds are scatter-plotted.

  Several hundred trials are present, so the "trial stride" argument is used to plot only the first trial out of every 25, for demonstration purposes.

Typical plots generated by this script are shown in Figures 3.3 through 3.6.

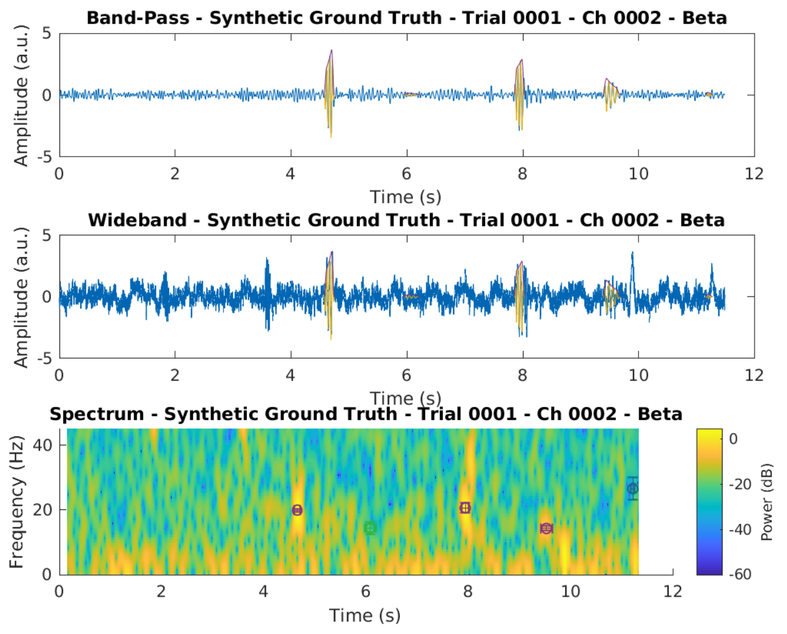


Figure 3.3: Synthetic data - typical ground truth events.

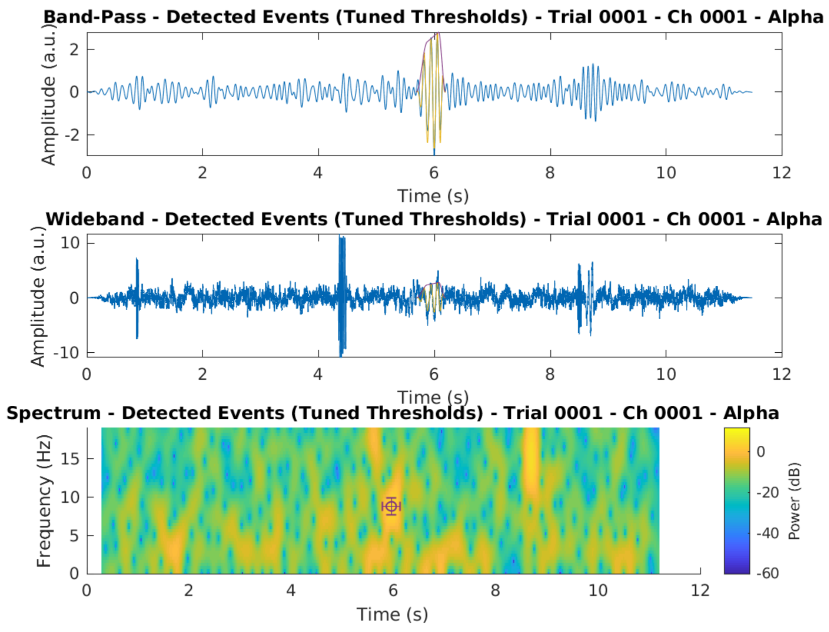


Figure 3.4: Synthetic data - typical detected events.

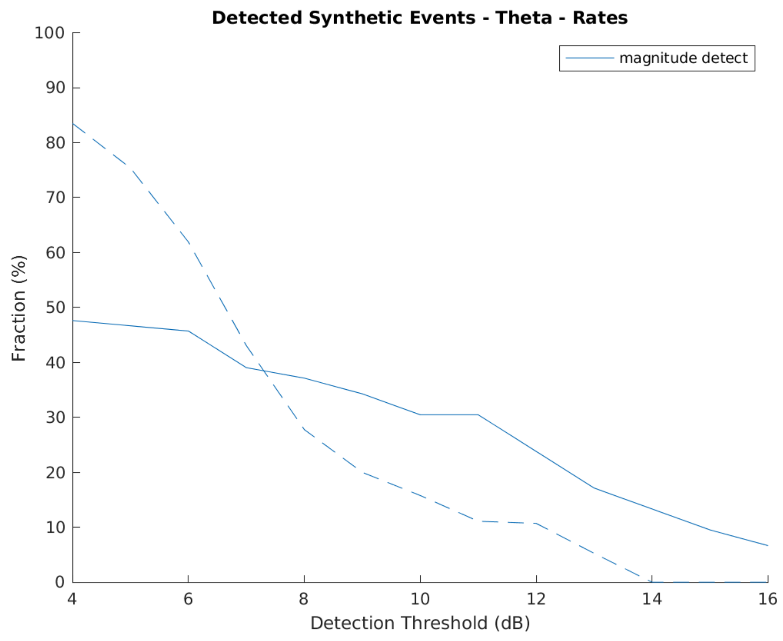


Figure 3.5: Synthetic data - typical detection rates vs threshold curve. The solid line is the fraction of ground truth events detected; the dashed line is the fraction of reported events that were false positives.

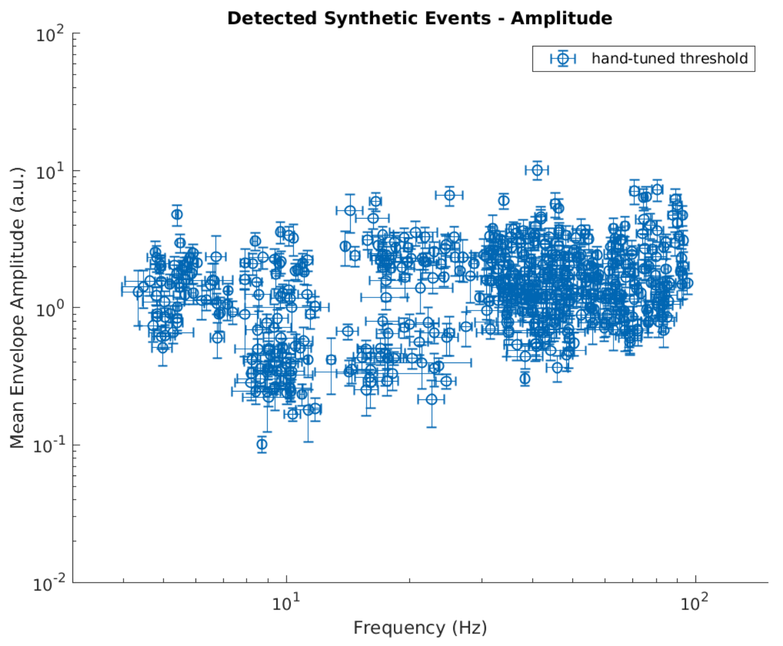


Figure 3.6: Synthetic data - scatter-plot of detected amplitude vs detected frequency.

3.4 Minimal Synthetic Data Processing Script

3.4.1 do_minimal_synth.m

```
% Minimal tutorial script - Comparing ground truth and detected burst events
% with synthetic data.
% See LICENSE.md for distribution information.


%
%
% Configuration.

%
% Paths.

addpath('lib-wl-synth', 'lib-wl-proc', 'lib-wl-ft', 'lib-wl-aux', ...
  'lib-wl-plot');


%
% Data files.

outputfile = 'output/minimal-synthevents.mat';
plotdir = 'output';


%
% Bands of interest.

% NOTE - Splitting gamma, as detection works best if bands are about one
% octave wide.
bandlist = [ ...
  struct('band', [ 4 7.5 ],    'label', 'th', 'name', 'Theta') ...
  struct('band', [ 7.5 12.5 ], 'label', 'al', 'name', 'Alpha') ...
  struct('band', [ 12.5, 30 ], 'label', 'be', 'name', 'Beta') ...
  struct('band', [ 30 60 ],    'label', 'gl', 'name', 'Gamma (low)') ...
  struct('band', [ 60 100 ],   'label', 'gh', 'name', 'Gamma (high)') ];


%
% Segmentation and parameter extraction.

% NOTE - The "3 sigma" rule of thumb is 9.5 dB.

% Detection by magnitude thresholding.

segconfig = struct( 'type', 'magdual', ...
  'qlong', 10, 'qdrop', 0.5, 'qglitch', 1.0, ...
  'dbpeak', 10, 'dbend', 2 );

% Envelope fitting by a coarse grid search.

paramconfig = struct( 'type', 'grid', 'gridsteps', 7 );


% Brute force threshold sweep range.
detsweepthresholds = 4:1:16;

% Per-band fine-tuned thresholds.
tunedthresholds = [ 8, 12, 12, 8, 9 ];


% The only override we have is DC averaging for theta.

for bidx = 1:length(bandlist)
  bandoverrides(bidx) = struct( 'seg', struct(), 'param', struct() );
end
bandoverrides(1).seg.qlong = inf;



%
%
% Data generation.


% Burst types.
% Min/max duration (periods), frequency ramping, amplitude ramping.
bpeep =  struct( 't', [1 2], 'f', [1   1  ], 'a', [1   1  ] );
bchirp = struct( 't', [2 4], 'f', [0.7 1.5], 'a', [0.3 3  ] );
btone =  struct( 't', [3 6], 'f', [0.8 1.2], 'a', [0.7 1.3] );

% Event generation specification.
% Each record has a band, a rate, and a burst definition.

% Theta band: peeps, 0.5/sec.
% Alpha band: tones (0.2/sec) and chirps, (0.4/sec).
% Beta band: chirps, 1/sec.
% Gamma band: tones, 2/sec.

% This is an array of structs; Matlab's syntax for building this is peculiar.

burstdefs = struct ( 'snrrange', [ -10 20 ], ...
  'rate',      { 0.5,     0.2,         0.4,         1,          2 }, ...
  'noiseband', { [4 7.5], [7.5 12.5], [7.5 12.5], [12.5 30], [30 100 ] }, ...
  'fctrrange', { [4 7.5], [7.5 12.5], [7.5 12.5], [12.5 30], [30 100 ] }, ...
  'durrange',  { bpeep.t, btone.t,    bchirp.t,   bchirp.t,  btone.t }, ...
  'framprange',{ bpeep.f, btone.f,    bchirp.f,   bchirp.f,  btone.f }, ...
  'aramprange',{ bpeep.a, btone.a,    bchirp.a,   bchirp.a,  btone.a } );


%
% Do the generation.

fsamp = 1000;

chancount = 6;
trialcount = 10;

% Trial duration is a range.
trialdur = [ 10 20 ];

% Variation range for event rates and SNR across channels.
channelratevar = [ 0.1 1.0 ];
channelnoisevar = [ -10 10 ];



disp('-- Generating synthetic data.');


% Generate the data traces and lumped ground truth.
[ synthdata synthgt_lumped ] = wlSynth_genFieldTrip( ...
  fsamp, chancount, trialcount, trialdur, ...
  burstdefs, channelratevar, channelnoisevar);

% Generate per-band ground truth.
synthgt_byband = wlAux_splitEvMatrixByBand(synthgt_lumped, bandlist);



%
%
% Perform a brute-force sweep of detection threshold.


% Wave error function, for pruning.
% This calculates the relative RMS error between the reconstructed event and
% the band-pass-filtered input waveform.

waveerrfunc = @(thisev, thiswave) ...
  wlProc_calcWaveErrorRelative( thiswave.bpfwave, thisev.sampstart, ...
    thisev.wave, thisev.s1 );

% Event filtering function, for pruning.
% This rejects anything with 'errbpf' above 0.7.
prunepassfunc = @(thisev) (0.7 >= thisev.auxdata.errbpf);
```

```
%
% Perform the detection sweep.

disp('-- Performing brute-force detection threshold sweep.');


clear synthdetect;

for thidx = 1:length(detsweepthresholds)

  % Make a copy of "band overrides", and set the desired threshold.

  bandoverrides_test = bandoverrides;
  for bidx = 1:length(bandlist)
    bandoverrides_test(bidx).seg.dbpeak = detsweepthresholds(thidx);
  end


  % Detect events, and trim events at the ends to avoid rolloff artifacts.

  detect_temp = wlFT_doFindEventsInTrials_MT( synthdata, bandlist, ...
    segconfig, paramconfig, bandoverrides_test, true );

  padtime = 0.5;
  detect_temp = wlFT_pruneEventsByTime(detect_temp, padtime, padtime);


  % Calculate reconstruction error and prune anything that's too large.

  detect_temp = wlFT_calcEventErrors( detect_temp, waveerrfunc, 'errbpf' );
  detect_temp = wlAux_pruneMatrix( detect_temp, prunepassfunc );


  % Store this result in our detection list.

  synthdetect{thidx} = detect_temp;

end


%
% Build an event matrix using the selected threshold from each band.

synthdetect_selected = synthdetect{1};

for bidx = 1:length(bandlist)

  threshdb = tunedthresholds(bidx);
  % Get the threshold sweep array index corresponding to this threshold.

  thcompare = abs(detsweepthresholds - threshdb) < 0.49;
  thidxlist = 1:length(detsweepthresholds);
  thidx = thidxlist(thcompare);


  % Copy the relevant slices from the full detection list.
  % Remember to use parentheses rather than curly braces for slices; we're
  % not dereferencing.

  synthdetect_selected.events(bidx,:,:) = ...
    synthdetect{thidx}.events(bidx,:,:);
  synthdetect_selected.waves(bidx,:,:) = ...
    synthdetect{thidx}.waves(bidx,:,:);
  synthdetect_selected.auxdata(bidx,:,:) = ...
    synthdetect{thidx}.auxdata(bidx,:,:);

  synthdetect_selected.segconfigbyband{bidx} = ...
    synthdetect{thidx}.segconfigbyband{bidx};
  synthdetect_selected.paramconfigbyband{bidx} = ...
    synthdetect{thidx}.paramconfigbyband{bidx};

end



%
%
% Save the synthetic data and raw detection results to disk.

disp('-- Saving synthetic data, ground truth, and detected events.');

% FIXME - This doesn't save "synthdetect", due to size issues.
% This is for good reason - it's 5 GB (due to waveform copies).

save( outputfile, 'synthdata', 'synthgt_lumped', 'synthgt_byband', ...
  'detsweepthresholds', 'synthdetect', ...
  'tunedthresholds', 'synthdetect_selected' );



%
%
% Calculate confusion matrix statistics.


disp('-- Calculating event statistics.');
```

```
% Event comparison function; this determines whether two event records
% "match" (refer to the same event).

matchfreq = 1.5;    % Worst-case frequency ratio.
matchamp = 3.0;     % Worst-case amplitude ratio.
matchlength = 4.0;  % Worst-case length ratio.
matcholap = 0.75;   % Worst-case fraction of the smaller event that's covered.

evcomparefunc = @(evfirst, evsecond) ...
  wlProc_calcMatchFromParams( evfirst, evsecond, ...
    matchfreq, matchamp, matchlength, matcholap );


% Compare against ground truth, and record the resulting statistics.
% Don't save this to disk; it's fast to compute, and huge.

clear confstats;

for thidx = 1:length(detsweepthresholds)

  [ fpmat fnmat tpmat matchtruth matchtest missingtruth missingtest ] = ...
    wlFT_compareMatrixEventsVsTruth( ...
      synthgt_byband, synthdetect{thidx}, evcomparefunc );

  confstats{thidx} = struct( ...
    'fp', fpmat, 'fn', fnmat, 'tp', tpmat, ...
    'truematch', matchtruth, 'truemissing', missingtruth, ...
    'testmatch', matchtest, 'testmissing', missingtest );

end



%
%
% Plot detected events.


disp('-- Generating plots.');


%
% Common configuration.

figconfig = struct( ...
  'fig', figure, 'outdir', plotdir, ...
  'fsamp', fsamp, ...
  'psfres', 5, 'psolap', 99, 'psleak', 0.75, 'psylim', 50 );

plots_per_band = inf;
```

```
trial_stride = 25;
channel_stride = 1;


%
% Merge the "selected" event matrix so that it can be scatter-plotted.

% This should never match.
evcomparefuncfalse = @(evfirst, evsecond) deal(false, inf);

% Merge across trials and channels.
synthdetect_merged = ...
  wlAux_mergeTrialsChannels(synthdetect_selected, evcomparefuncfalse);

% Merge across bands.
bandlistwide = [ struct( 'band', [ 2 200 ], 'label', 'wb', 'name', 'Wide' ) ];
synthdetect_merged = ...
  wlAux_splitEvMatrixByBand(synthdetect_merged, bandlistwide);


%
% Event plots.


% Ground truth events.

wlPlot_plotAllMatrixEvents(figconfig, synthgt_byband, ...
  'Synthetic Ground Truth', 'synthgt', ...
  plots_per_band, trial_stride, channel_stride);


% Detected events for selected thresholds.

wlPlot_plotAllMatrixEvents(figconfig, synthdetect_selected, ...
  'Detected Events (Tuned Thresholds)', 'synthdet', ...
  plots_per_band, trial_stride, channel_stride);


%
% Detection rates vs. threshold.

for bidx = 1:length(bandlist)

  chartfp = [];
  chartfn = [];
  charttp = [];

  for thidx = 1:length(detsweepthresholds)

    thisconf = confstats{thidx};
    chartfp(thidx) = sum(sum( thisconf.fp(bidx,:,:) ));
    chartfn(thidx) = sum(sum( thisconf.fn(bidx,:,:) ));
    charttp(thidx) = sum(sum( thisconf.tp(bidx,:,:) ));

  end

  wlPlot_plotDetectCurves( figconfig, ...
    detsweepthresholds, 'Detection Threshold (dB)', ...
    [ struct( 'fp', chartfp, 'fn', chartfn, 'tp', charttp, ...
        'color', [ 0.0 0.4 0.7 ], 'label', 'magnitude detect' ) ], ...
    sprintf('Detected Synthetic Events - %s', bandlist(bidx).name), ...
    sprintf('thresh-%s', bandlist(bidx).label) );

end


%
% Scatter-plots of detected event parameters.

plotband = [ 3 150 ];
plotdur = [ 0 15 ];
plotamp = [ 1e-2 100 ];
wlPlot_plotEventScatterMulti( figconfig, ...
  [ struct( 'eventlist', synthdetect_merged.events{1}, ...
    'color', [ 0.0 0.4 0.7 ], 'legend', 'hand-tuned threshold') ], ...
  plotband, plotdur, plotamp, 'Detected Synthetic Events', 'synthdet' );


%
% There's a lot more that _can_ be plotted, but this example is long enough.
% See "do_ft_synth_plot.m" for more examples.



%
%
% Done.


disp('-- Done.');



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

Appendix A

Special Structures and Function Handles

A.1 BURSTTYPEDEF.txt

```
wlSynth_traceAddBursts() generates oscillatory bursts with randomly varied
parameters. The range of possible values for these parameters is specified
by "burst type definition" structures, with the following fields:

"rate":                    Average number of bursts per second.
"snrrange": [min max]      Signal-to-noise ratio of bursts, in dB.
"noiseband": [min max]     Frequency band in which noise power is measured.
"durrange": [min max]      Burst duration in cycles (at average frequency).
"fctrrange": [min max]     Burst nominal center frequency.
"framprange": [min max]    Ratio of ending/starting frequency.
"aramprange": [min max]    Ratio of ending/starting amplitude.

Burst events of a given type are treated as a Poisson point process with the
specified rate. For each event, parameters are chosen randomly within the
specified ranges.

SNR in decibels is drawn using a uniform distribution. All other parameters
are drawn using a log distribution (the logarithm of the parameter values
has a uniform distribution). This handles diverse scales well.
```

A.2 COMPAREFUNC.txt

```
A "comparison function" is used for determining whether two oscillatory
burst event descriptions refer to the same event or different events, and
to meansure the "distance" between oscillatory burst events in parameter
space.

The implementation of this function is arbitrary; comparison functions are
passed as "lambda functions" (functiton handles) to library functions that
```

```
need them.

An event comparison function has the form:

[ ismatch distance ] = comparefunc(evfirst, evsecond)

The value of "ismatch" should be boolean.
The value of "distance" should be a non-negative scalar.
```

A.3 EVENTFORMAT.txt

```
Events are described by structures with the following fields. Some of these
may be omitted, depending on the function producing the event list:


"sampstart":  Sample index in the original recording waveform corresponding
              to burst time 0.
"duration":   Time between nominal 50% roll-on and 50% roll-off for the burst.

"s1":         Sample index in "wave" of nominal start (time 0, 50% of roll-on).
"s2":         Sample index in "wave" of nominal stop (50% of roll-off).
"samprate":   Samples per second in the stored waveforms.

"snr":        Nominal signal-to-noise ratio for this burst, in dB.

Curve-fit parameters:

"paramtype":  Type of parameter fit performed. For now, "chirpramp".

"f1":         Burst frequency at nominal start.
"f2":         Burst frequency at nominal stop.
"a1":         Envelope amplitude at nominal start.
"a2":         Envelope amplitude at nominal stop.
"p1":         Phase at nominal start.
"p2":         Phase at nominal stop.

"rollon":     Duration of the envelope's curve-fit cosine roll-on time.
"rolloff":    Duration of the envelope's curve-fit cosine roll-off time.
"ftype":      Frequency ramp type ("linear" or "logarithmic").
"atype":      Amplitude ramp type ("linear" or "logarithmic").

Curve-fit waveform:

"times":      [1xN] array of burst sample times, relative to start time.
"wave":       [1xN] array of nominal (curve-fit) burst waveform samples.
"mag":        [1xN] array of nominal envelope magnitude samples.
"freq":       [1xN] array of nominal instantaneous frequency samples.
"phase":      [1xN] array of nominal instantaneous phase samples (in radians).
```

```
Auxiliary waveforms (optional; typically algorithm-specific):

"auxwaves":   Structure containing fields that each contain a signal.

  Signal labels are arbitrary but typically have the following forms:

  "FOOtimes":    [1xN] array with signal FOO's sample times (relative).
  "FOOwave":     [1xN] array with signal FOO's waveform samples.
  "FOOmag":      [1xN] array with signal FOO's instantaneous magnitude.
  "FOOfreq":     [1xN] array with signal FOO's instantaneous frequency.
  "FOOphase" :   [1xN] array with signal FOO's instantaneous phase.

Auxiliary metadata (optional; typically algorithm-specific):

"auxdata":   Structure containing algorithm-specific event metadata.

  Metadata labels are arbitrary but typically have the following form:

  "FOOstat":     Statistic "stat" for algorithm/processing step "FOO".
```

A.4 EVMATRIX.txt

```
The "event matrix" structure holds event information extracted from Field
Trip data. This data is kept in wlBurst library format, to avoid discarding
metadata and diagnostic information. This may be converted to and from
the "event FT" format, with the loss of some metadata.

An "event matrix" structure contains the following fields, which hold data
returned by "wlProc_doSegmentAndParamExtract":

- "events{bidx, tidx, cidx}" holds event lists corresponding to a given
  band, FT trial, and channel within the FT trial.

- "waves{bidx, tidx, cidx}" holds diagnostic waveform information from a
  given band, FT trial, and channel within the FT trial. This includes
  raw FT trial waveforms ("ftwave") and FT timestamps ("fttimes").

- "auxdata{bidx, tidx, cidx}" holds auxiliary diagnostic data from a given
  band, FT trial, and channel within the FT trial.

The following additional fields are also present:

- "bandinfo" is an array of structures with the following fields:
  "band" [ min max ] defines a frequency band of interest.
  "name" is a character array containing a human-readable band name.
  "label" is a character array containing a filename-safe band ID string.
```

```
- "samprate" is the sampling rate (samples per second).

- "segconfigbyband{bidx}" records segmentation algorithm information per
  "SEGCONFIG.txt".

- "paramconfigbyband{bidx}" records parameter estimation algorithm
  information per "PARAMCONFIG.txt".
```

A.5 FIGCONFIG.txt

```
The "figure configuration information" structure contains the following
fields:

General parameters:

"fig" - A figure handle to perform rendering on.
"outdir" - The directory to write figure files to.

"fsamp" - Time series sampling rate. Used for spectrum plots.

Spectrum plot tuning parameters [typical values in square brackets]:

"psfres" - Frequency resolution, Hz. [20]
"psolap" - Overlap between time and frequency bins, percent. [99]
"psleak" - "Leakage" between bins (0.5 = Hann window, 1.0 = square).
  [0.75]
"psylim" - Maximum frequency plotted. [50]



Tuning time/frequency plots is a black art. A useful introduction is at:
https://www.mathworks.com/help/signal/examples/practical-introduction-to-time-frequency-analysis.h

Details of the implementation of the "pspectrum" function are at:
https://www.mathworks.com/help/signal/ref/pspectrum.html
```

A.6 PARAMCONFIG.txt

```
Parameter extraction algorithm configuration structures have a mandatory
field "type", with additional fields that depend on the algorithm specified
by the "type" field.

Algorithm-specific arguments are as follows:
```

```
== "groundtruth" type:

Parameters are known a priori.



== "default" type:

The wlProc library is to replace this with a configuration of its choice.
FIXME - NYI.



== "custom" type:

This wraps a user-specified parameter extraction function.

"paramfunc":  A function handle with the form:
  [ newevents, auxdata ] = paramfunc( oldevents, waves, samprate, paramconfig )

The function handle is passed a copy of this structure as its "paramconfig"
argument, allowing additional parameters to be passed.

Function arguments are:
  "oldevents":  The list of events produced by segmentation. Only the
    "sampstart" and "duration" fields are guaranteed to exist.
  "waves":  The list of derived waveforms produced by segmentation. Which
    of these exist depends on the segmentation algorithm. These typically
    include a band-pass-filtered waveform and its Hilbert transform
    magnitude, phase, and frequency series.
  "samprate":  The number of samples per second in the signal data.
  "paramconfig":  A copy of this structure, containing additional parameters.

Function return values are:
  "newevents":  An updated list of events with remaining fields filled in.
  "auxdata":  A structure containing additional statistics or diagnostic
    data. This structure may be empty.



== "grid" type:

Amplitude envelope roll-on and roll-off are fit using a coarse grid search.
For a given roll-on and roll-off, magnitude and frequency are curve-fit.
Endpoints are kept fixed.

"gridsteps":  The number of intermediate time points tested as roll-on and
              roll-off endpoints.

The number of cases tested goes up as the square of "gridsteps".
```

```
== "annealamp" type:

Perform "grid" for a coarse fit, then use simulated annealing on the envelope
to fit it to the analytic signal magnitude signal.

"gridsteps":   The number of intermediate time points for the "grid" fit.
"matchfreq":   The maximum allowed frequency ratio between original and
               perturbed events.
"matchamp:     The maximum allowed peak-amplitude ratio between original and
               perturbed events.
"matchlength": The maximum allowed length ratio between original and perturbed
               events.
"matcholap":   The minimum fraction of the shorter event that must be covered
               by the larger event, between original and perturbed events.
"tunnelmax":   The maximum number of consecutive unproductive perturbation
               attempts that can be made before annealing is assumed to have
               converged.
"totalmax":    The maximum number of perturbation attempts that can be made
               in total while annealing one event record.


== "annealboth" type:

Perform "grid" for a coarse fit, then use simulated annealing on the envelope
to fit it to the analytic signal magnitude signal, then use simulated
annealing on amplitude and frequency to fit the event and signal waveforms.

Arguments are identical to "annealamp".



This is the end of the file.
```

A.7 SEGCONFIG.txt

```
Segmentation algorithm configuration structures have a mandatory field
"type", with additional fields that depend on the algorithm specified by the
"type" field.

Algorithm-specific arguments are as follows:



== "groundtruth" type:

Segments are known a priori.
```

```
== "default" type:

The wlProc library is to replace this with a configuration of its choice.
FIXME - NYI.



== "custom" type:

This wraps a user-specified segmentation function.

"segmentfunc":  A function handle with the form:
  [ events, waves ] = segmentfunc( wavedata, samprate, bpfband, segconfig )

The function handle is passed a copy of this structure as its "segconfig"
argument, allowing additional parameters to be passed.

Function arguments are:
  "wavedata":  The data trace to examine.
  "samprate":  The number of samples per second in the signal data.
  "bpfband" [min max]:  The frequency band of interest. Edges may fade.
  "segconfig":  A copy of this structure, containing additional parameters.

Function return values are:
  "events":  An array of event record structures per EVENTFORMAT.txt.
    Only the "sampstart" and "duration" fields are provided.
  "waves":   A structure containing new waveforms derived from "wavedata".
    These typically include a band-pass-filtered waveform and its Hilbert
    transform magnitude, phase, and frequency series.


== "mag" type:

Detection looking for excursions in analytic signal magnitude.
As "magdual", but omitting "dbend".



== "magdual" type:

Detection looking for excursions in analytic signal magnitude.

"qlong":    Time constant for DC-average filtering, as a multiple of the
            nominal oscillation period.
"qdrop":    Maximum gap in detection to ignore, as a multiple of the nominal
            oscillation period.
"qglitch":  Maximum spurious detection to ignore, as a multiple of the nominal
```

```
            oscillation period.
"dbpeak":   Minimum power ratio (in dB) above background for event detection.
"dbend":    Minimum power ratio (in dB) above background at the edges of a
            detected event. This is used to determine event boundaries.

The standard "3 sigma" heuristic for detection corresponds to a "dbpeak"
value of 9.5 dB. A typical "dbend" value is 2 dB.



== "freq" type:

Detection looking for stabilization of analytic signal frequency.
As "freqdual", but omitting "dbend".



== "freqdual" type:

Detection looking for stabilization of analytic signal frequency.

This involves adding high-frequency noise to the band-pass-filtered signal;
oscillations with amplitude significantly higher than the HF noise floor
stabilize the analytic signal's computed frequency (derivative of phase).

"noiseband":  [min max]  Frequency band for noise injection.
"noisesnr":   Ratio of signal power to injected noise power (in dB).
"qlong":    Time constant for DC-average filtering, as a multiple of the
            nominal oscillation period.
"qshort":   Time constant for smoothing analytic signal frequency, as a
            multiple of the nominal oscillation period.
"qdrop":    Maximum gap in detection to ignore, as a multiple of the nominal
            oscillation period.
"qglitch":  Maximum spurious detection to ignore, as a multiple of the nominal
            oscillation period.
"dbpeak":   Minimum noise power suppression ratio (in dB) for event detection.
"dbend":    Minimum noise power suppression ratio (in dB) at the edges of a
            detected event. This is used to determine event boundaries.

The noise band's low frequency edge is typically at least 10 times the event
band's upper frequency edge.
The standard "3 sigma" heuristic for detection corresponds to a "dbpeak"
value of 9.5 dB. A typical "dbend" value is 2 dB.



This is the end of the file.
```

A.8 WAVEERRFUNC.txt

```
A "wave error function" is used for evaluating the "error" (distance) between
an event and a reference waveform. Typically this is done via time-domain
comparison between the event waveform and reference waveform, but other
approaches are possible.

The implementation of this function is arbitrary; wave error functions are
passed as "lambda functions" (function handles) to library functions that
need them.


A wave error function has the form:

error = errfunc(event, wavestruct)

The value of "error" should be a non-negative scalar.

"event" is an event record, per EVENTFORMAT.txt.
"wavestruct" is a structure containing one or more reference waveforms. This
  is typically one cell extracted from the "waves" field of an event matrix,
  per EVMATRIX.txt.


An example implementation is:

waveerrfunc_bpf = @(thisev, thiswave) ...
  wlProc_calcWaveErrorRelative( thiswave.bpfwave, thisev.sampstart, ...
    thisev.wave, thisev.s1 );
```

A.9 WLNOTES.txt

```
In addition to the raw field trip structures, the wlFT wrapper routines
produce a "wlnotes" structure containing additional metadata. Contents
are described below.


"wlnotes.bandinfo" is an array of structures containing band definitions:

- "band" [ min max ] contains the frequency limits of the band.
- "name" is a character array containing a human-readable band name.
- "label" is a unique filename-safe character string identifying the band.


"wlnotes.wavesbybandandtrial{bidx,tidx,cidx}" is a cell array containing
structures that contain diagnostic waveforms. These waveforms are derived
from the original trial data and represent intermediate processing steps.
```

```
Different analyses will produce different sets of diagnostic waveforms.

wlBurst event records that are annotated with "context" typically have
corresponding portions of these signals in their "auxwaves" structures.
wlBurst event matrices store these in the "waves" cell array.

- "ftwave" is the original wideband trial waveform.
- "fttimes" contains original timestamps associated with "ftwave" data.

- "bpfwave" is the band-pass-filtered trial waveform.
- "bpf(mag|freq|phase)" are analytic signals derived from "bpfwave", from
  analyses that use the Hilbert transform.

- "magpower(fast|slow)" are low-pass-filtered squared magnitudes computed
  from the analytic magnitude of the signal, from analyses that use
  magnitude-based detection.

- "noisywave" is a copy of "bpfwave" with high-frequency noise added, from
  analyses that use frequency-stability detection.
- "noisy(mag|freq|phase)" are analytic signals derived from "noisywave",
  from analyses that use frequency-stability detection.
- "fvar(fast|slow)" are low-pass-filtered versions of the squared magnitude
  of the AC component of "noisyfreq". These represent the instantaneous
  variance of "noisyfreq" averaged over narrow and wide time windows.


"wlnotes.trialinfo_label" is a cell array of character arrays containing
labels for columns in the "trialinfo" metadata structure. Metadata present
depends on the analyses performed.

wlBurst event records typically store some or all of this metadata in their
"auxdata" structures (aside from curve-fit event parameters, which are
stored as top-level event record fields). wlBurst event matrices store this
in the "auxdata" cell array. In wlBurst event and event matrix structures,
the string "ft_" is prepended to the labels associated with these fields
(i.e. "ft_sampstart" in auxdata, vs "sampstart" in wlnotes).

Metadata that is always present:

- "sampstart" is a duplicate of "sampleinfo(trial,1)".
- "sampend" is a duplicate of "sampleinfo(trial,2)".
- "trialnum" is the original-data trial in which this event occurred.
- "channelnum" is the original-data channel in which this event occurred.
- "bandnum" is the index of the detection band within "bandinfo".
- "bandmin" is a duplicate of "bandinfo.band(1)".
- "bandmax" is a duplicate of "bandinfo.band(2)".
- "eventnum" is the event number within this trial, channel, and band.

Event detection metadata:
```

- "detthresh" is the minimum peak power or variance excursion for event detection, in dB.
- "trial_bpf_rms" is the RMS amplitude of the band-pass-filtered version of the original trial ("bpfwave" in "wavesbybandandtrial").
- "event_rms" is the RMS amplitude of the reconstructed event.
- "event_max" is the maximum absolute amplitude of the reconstructed event.

Cropped trial metadata:

- "roistart" is the first copied sample location in the un-cropped trial.
- "roistop" is the last copied sample location in the un-cropped trial.

Parameter fit metadata:

- "f1" is the curve-fit frequency at 50% roll-on.
- "f2" is the curve-fit frequency at 50% roll-off.
- "a1" is the curve-fit envelope magnitude at 50% roll-on.
- "a2" is the curve-fit envelope magnitude at 50% roll-off.
- "p1" is the curve-fit phase at 50% roll-on.
- "p2" is the curve-fit phase at 50% roll-off.
- "rollon" is the duration of roll-on, in seconds.
- "rolloff" is the duration of roll-off, in seconds.
- "duration" is the time between 50% roll-on and 50% roll-off, in seconds.

"wlnotes.segconfigbyband{bidx}" and "wlnotes.paramconfigbyband{bidx}" store the algorithm tuning parameters used for segmentation and parameter extraction of events in the various frequency bands, respectively.

Appendix B

Sample Code – Configuration

B.1 do_ft_init.m

```
% Field Trip-related test scripts - Initialization.

% Paths.

addpath('lib-wl-synth');
addpath('lib-wl-proc');
addpath('lib-wl-ft');
addpath('lib-wl-aux');
addpath('lib-wl-plot');


% Colours.
% Mostly cribbed from get(gca,'colororder') (the defaults).

cblu = [ 0.0 0.4 0.7 ];
cbrn = [ 0.8 0.4 0.1 ]; % Tweaked; original was [ 0.9 0.3 0.1 ].
cyel = [ 0.9 0.7 0.1 ];
cmag = [ 0.5 0.2 0.5 ];
cgrn = [ 0.5 0.7 0.2 ];
ccyn = [ 0.3 0.7 0.9 ];
cred = [ 0.6 0.1 0.2 ];


%
% Signal information.

datadir = 'data';

fsamp = 1000;

bandwide = [ 1 400 ];
```

```
bandtheta = [ 4 7.5 ];
bandalpha = [ 7.5 12.5 ];
bandbeta = [ 12.5 30 ];
bandgammalo = [ 30 60 ];
bandgammahi = [ 60 100 ];

bandgamma = [ bandgammalo(1) bandgammahi(2) ];

bandlist = [ ...
  struct('band', bandtheta,   'label', 'th', 'name', 'Theta') ...
  struct('band', bandalpha,   'label', 'al', 'name', 'Alpha') ...
  struct('band', bandbeta,    'label', 'be', 'name', 'Beta') ...
  struct('band', bandgammalo, 'label', 'gl', 'name', 'Gamma (Low)') ...
  struct('band', bandgammahi, 'label', 'gh', 'name', 'Gamma (high)') ];

bandlistbands = { bandtheta, bandalpha, bandbeta, bandgammalo, bandgammahi };

bandlistwide = ...
  [ struct('band', bandwide, 'label', 'wb', 'name', 'Wideband') ];


%
% Synthetic Field Trip data parameters.

synth_use_small_dataset = false;

% NOTE - This switch affects filenames too, so that both versions can be
% saved/loaded as needed.
synth_use_one_trial = false;


synth_trialcount = 20;
synth_chancount = 16;

% FIXME - Short trials have pretty bad edge artifacts.
%synth_trialdur = [ 3 5 ];
synth_trialdur = [ 5 8 ];

if synth_use_small_dataset
  % FIXME - Use a smaller set for testing.
  synth_trialcount = 10;
  synth_chancount = 4;
end

synth_channelratevar = [ 0.1 1.0 ];
synth_channelnoisevar = [ -10 10 ];

synth_fname_data = sprintf('%s/ftsynth.mat', datadir);
synth_fname_detect = sprintf('%s/ftsynthdetect.mat', datadir);
```

```
if synth_use_one_trial

  synth_trialdur = synth_trialdur * synth_trialcount;
  synth_trialcount = 1;

  synth_fname_data = sprintf('%s/ftsynth-long.mat', datadir);
  synth_fname_detect = sprintf('%s/ftsynthdetect-long.mat', datadir);

end


%
% Default event segmentation information.

% Frequency-based detection: add noise and look for flat analytic frequency.

segbandnoise = [ 200 400 ];
segnoisesnrchosen = 10;  % Fixed noise SNR for frequency-based detection.

% NOTE - Making the "ac" time constant longer requires making the dropout
% period longer as well, as dropout timescale is set by smoothing time.

segfreqvarqlong = 10;  % Number of mid-frequency periods for "dc" filtering.
segfreqvarqshort = 0.25;  % Number of mid-frequency periods for "ac" smoothing.

segfreqglitchq = 1.0;  % Events shorter than this many periods are dropped.
segfreqdropoutq = 0.5;  % Gaps shorter than this many periods are ignored.


% Amplitude-based detection: threshold the BPF analytic magnitude.

segmagvarqlong = 10;  % Number of mid-frequency periods for "dc" filtering.

segmagglitchq = 1.0;  % Events shorter than this many periods are dropped.
segmagdropoutq = 0.5;  % Gaps shorter than this many periods are ignored.


% Detection threshold sweep information.
% Thresholds are in dB, comparing a quantity to its quasi-DC average.
% For magnitude detection, it's analytic magnitude; for frequency detection,
% it's the approximate variance of the analytic frequency.

% Endpoint-detection thresholds.
% Event-detection thresholds should exceed this.
segdetendthresh = 2;

% dB. Ratio of threshold vs average magnitude power.
segdetsweepspan = [ 4 20 ];
```

```
% Interval for sweeping detection thresholds.
segdetsweepstep = 1;

% Actual detection threshold sweep values.
% Starting value and step size are guaranteed. Last value will be less than
% or equal to the specified maximum.

detsweepthresholds = segdetsweepspan(1):segdetsweepstep:segdetsweepspan(2);


% Pre-chosen detection threshold values.
% NOTE - These should map to valid values in det(mag/freq)thresholds.

% Criteria for choosing "good" threshold values.
eventerrmax = 0.3;

% Hardcoded values; these can be overridden.

if synth_use_one_trial

  % Long-trace versions, with "RMS error <= 0.7" pruning.
  magthreshdb =  [ 10 12 12  8 10 ];
  freqthreshdb = [  8  8 10 10 14 ];

else

  % Short-trace versions, with "RMS error <= 0.7" pruning.
  magthreshdb =  [  8 16 12 12 10 ];
  freqthreshdb = [  2  6  6 10 12 ];

end


% Threshold range for plotting.
% These are relative to the chosen "good" thresholds.
% NOTE - If the middle value _isn't_ zero, we need to tweak thresholds.
plotthreshdbmag =  [ -4 0 4 ];
plotthreshdbfreq = [ -2 0 2 ];
plotthreshcolors = { cyel cbrn cblu };
plotthreshnames = { 'low', 'medium', 'high' };



% Ground truth matching parameters.

matchfreq = 1.5;    % Worst-case frequency ratio.
matchamp = 3.0;     % Worst-case amplitude ratio.
matchlength = 4.0;  % Worst-case length ratio.
matcholap = 0.75;   % Worst-case overlap fraction.
```

```
% Event binning for ground truth matching.

eventbintime = 5;    % Seconds per bin.
eventbinsearch = 1;  % Bins to search on either side for matches.

% Lambda function for match comparison.
evcomparefunc = @(evfirst, evsecond) ...
  wlProc_calcMatchFromParams(evfirst, evsecond, ...
    matchfreq, matchamp, matchlength, matcholap);

% Lambda function for "never match".
evcomparefuncfalse = @(evfirst, evsecond) deal(false, inf);

% Lambda function for "match only if really close".
% This is intended for merging events that were duplicated.
evcomparefuncdup = @(evfirst, evsecond) ...
  wlProc_calcMatchFromParams(evfirst, evsecond, ...
    matchfreq^0.2, matchamp^0.2, matchlength^0.2, matcholap^0.2);



%
% Curve-fitting information.

% Simulated annealing parameters.
annealmaxtries = 100;
%annealmaxtotal = 1000;
% NOTE - It takes about 10,000 steps for things to look really good.
annealmaxtotal = 10000;


%
% Figure information.

outdir = 'output';
scratchfig = figure;


% Context window around detected bursts.
% It's the greater of N cycles or N times the burst length on each side.
contextcycles = 4.0;
contextlengths = 2.0;


% Figure configuration structure.

figconfig.fig = scratchfig;
figconfig.outdir = outdir;

% Power spectrum parameters.
% NOTE - This is sensitive to the band we're looking at. These defaults
% are tolerable most of the time.
figconfig.fsamp = fsamp;
figconfig.psfres = 5;
figconfig.psolap = 99;
figconfig.psleak = 0.75;
figconfig.psylim = 50;


% Turn certain plot subsets on or off.

% Plot every Nth event.
eventstride = 25;

% Only emit the first K plots for any given band.
eventplotsperband = 4;

% Event matrix plotting configuration.

%bandplots_max = eventplotsperband;
bandplots_max = inf;

%stride_trial = eventstride;
stride_trial = 1;

stride_channel = 1;


%
% Reload data.


% Synthetic waveforms and ground truth.

if isfile(synth_fname_data) && (~exist('traceft', 'var'))
  disp('-- Loading synthetic FT data.');
  % Load everything; only relevant structures should be present.
  load(synth_fname_data);
  disp('-- Finished loading.');
end


% Detection results for synthetic events.

if isfile(synth_fname_detect) && (~exist('detectft_mag', 'var'))
  disp('-- Loading synthetic FT detection results.');
  % Load everything; only relevant structures should be present.
  load(synth_fname_detect);
  disp('-- Finished loading.');
end
```

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

B.2 do_ft_synth_config.m

```
% Field Trip-related test scripts - Synthetic FT data - Configuration.

%
%
% Includes.

% NOTE - Assume the parent called "init" already.



%
%
% Configuration.



%
% Event detection.


% Select the parameter extraction algorithm we want.
% "coarse" gives a poor fit, "amp" takes a while but gives a good fit, and
% "both" takes a very long time and gives a slightly better fit than "amp".

%param_chosen_type = 'coarse';
%param_chosen_type = 'amp';
param_chosen_type = 'both';

% Use DC average for theta thresholds.
seg_use_dc = [ true false false false false ];

% Padding, to ignore roll-off events.
padtime = 0.50;



%
% Plotting.


% Select which plots we want to generate.
```

```
plot_ftsyn_gt_band = true;
plot_ftsyn_gt_wide = false;

plot_ftsyn_det_waves = true;
plot_ftsyn_det_rates = true;
plot_ftsyn_det_scatter = true;

plot_ftsyn_raw_error = false;
plot_ftsyn_pruned_error = true;



%
% Event detection details.

% Detection.
% NOTE - We’ll be overriding "dbpeak".

segconfig_mag = struct( ’type’, ’magdual’, ...
  ’qlong’, segmagvarqlong, ...
  ’qdrop’, segmagdropoutq, ’qglitch’, segmagglitchq, ...
  ’dbpeak’, 10, ’dbend’, segdetendthresh );

segconfig_freq = struct( ’type’, ’freqdual’, ...
  ’noiseband’, segbandnoise, ’noisesnr’, segnoisesnrchosen, ...
  ’qlong’, segfreqvarqlong, ’qshort’, segfreqvarqshort, ...
  ’qdrop’, segfreqdropoutq, ’qglitch’, segfreqglitchq, ...
  ’dbpeak’, 10, ’dbend’, segdetendthresh );

% Parameter extraction.

paramconfig_coarse = struct( ’type’, ’grid’, ’gridsteps’, 7 );

paramconfig_amp = struct( ’type’, ’annealamp’, ’gridsteps’, 5, ...
  ’matchfreq’, matchfreq, ’matchamp’, matchamp, ...
  ’matchlength’, matchlength, ’matcholap’, matcholap, ...
  ’tunnelmax’, annealmaxtries, ’totalmax’, annealmaxtotal );

% This takes longer to anneal, so double the allowed number of steps.
paramconfig_both = struct( ’type’, ’annealboth’, ’gridsteps’, 5, ...
  ’matchfreq’, matchfreq, ’matchamp’, matchamp, ...
  ’matchlength’, matchlength, ’matcholap’, matcholap, ...
  ’tunnelmax’, annealmaxtries, ’totalmax’, 2 * annealmaxtotal );



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

Appendix C

Sample Code – Field Trip Recording

C.1 do_ft_thilo.m

```
% Field Trip-related test scripts - Extracting bursts from FT data.
%
% See LICENSE.md for distribution information.


%
%
% Includes.

do_ft_init



%
%
% Configuration.


% Data reduction, for fast tests.

use_one_band = false;
%one_band_chosen = 1;  % Theta.
%one_band_chosen = 2;  % Alpha.
%one_band_chosen = 3;  % Beta.
one_band_chosen = 4;  % Low gamma.
%one_band_chosen = 5;  % High gamma.


% Plotting decimation and limits.
% NOTE - Decimating trials, not events within a trial.
```

```
stride_trial = eventstride;
%stride_trial = 1;

stride_channel = 1;

%bandplots_max = eventplotsperband;
bandplots_max = inf;


% Filtering.

% 60 Hz notch filter for input waveforms.
use_notch = true;
notch_center = 60.0;
notch_halfwidth = 3.0;
notch_order = 20;


% Data.

inputfile = 'thilo-data/LFP_B_tmplate_01.mat';
inputvar = 'DATA';
targetchannels = { 'CSC5LFP' };
outputfile = sprintf('%s/ftthiloevents.mat', datadir);


% Manually specified padding for Thilo's data.
% Startup has detection-stabilization artifacts due to filtering.
% Reward has artifacts due to monkey anticipation motions disturbing probes.
% Frequency detection also gets trains of spurious events near the end.

padtimestart = 0.40;
padtimeend = 0.40;

% This flag sets whether we load Thilo-specific data for reward times.
% The fallback is to do artifact detection instead.

use_thilo_specific = true;


% Region of interest function - fallback version.
% NOTE - Using "deal" to return multiple outputs.

% ROI is the entire trial.
cropfunc = @(thistrial, sampinforow, trialinforow) ...
  deal( 1, (1 + sampinforow(2) - sampinforow(1)) );

% FIXME - Test version, with the ends clipped for debugging.
%cropfunc = @(thistrial, sampinforow, trialinforow) ...
%  deal( 1 + 50, (1 + sampinforow(2) - sampinforow(1)) - 80 );
```

```
%
% Segmentation and extraction.

% Set up defaults that are appropriate for the curated FT data.
% These can be overridden by band-specific values.

% NOTE - The "3 sigma" rule of thumb is 9.5 dB.
% If dbpeak is less than dbend, the original detection segments are used
% as-is rather than being extended.


% NOTE - Global values get overridden by band-specific values.

segconfig_mag = struct( 'type', 'magdual', ...
  'qlong', 10, ...
  'qdrop', 1.0, 'qglitch', 1.0, ...
  'dbpeak', 6, 'dbend', 2 );

segconfig_freq = struct( 'type', 'freqdual', ...
  'noiseband', segbandnoise, 'noisesnr', segnoisesnrchosen, ...
  'qlong', 10, 'qshort', 0.25, ...
  'qdrop', 1.0, 'qglitch', 0.5, ...
  'dbpeak', 3, 'dbend', 2 );


paramconfig_coarse = struct( 'type', 'grid', 'gridsteps', 7 );

paramconfig_anneal = struct( 'type', 'annealamp', 'gridsteps', 5, ...
  'matchfreq', matchfreq, 'matchamp', matchamp, ...
  'matchlength', matchlength, 'matcholap', matcholap, ...
  'tunnelmax', annealmaxtries, 'totalmax', annealmaxtotal );

% Two-step annealing with twice the maximum step count.
% This will hopefully converge most of the time.
paramconfig_annealfull = struct( 'type', 'annealboth', 'gridsteps', 5, ...
  'matchfreq', matchfreq, 'matchamp', matchamp, ...
  'matchlength', matchlength, 'matcholap', matcholap, ...
  'tunnelmax', annealmaxtries, 'totalmax', annealmaxtotal * 2 );

% The actual configurations to be used.

%paramconfig_chosen = paramconfig_coarse;
%paramconfig_chosen = paramconfig_anneal;
paramconfig_chosen = paramconfig_annealfull;


% Band-specific parameter values.
```

```
dbpeak_mag = [ 6, 7, 6, 5, 4 ];
dbend_mag =  [ 5, 2, 1, 1, 0 ];
%qlong_mag =  [ 10, 10, 10, 10, 30 ];
qlong_mag =  [ inf, 10, 10, 10, inf ];  % Use DC average for theta, gamma_hi
qdrop_mag =  [ 0.25, 0.5, 0.5, 1.0, 3.0 ];

dbpeak_freq = [ 0, 3, 3, 4, 4 ];
dbend_freq =  [ -1, 2, 1, 1, 0 ];
qlong_freq =  [ 10, 10, 10, 10, 30 ];
%qlong_freq =  [ inf, 10, 10, 10, inf ];  % Use DC average for theta, gamma_hi
qshort_freq =  [ 0.25, 0.25, 0.25, 0.5, 1.0 ];
qdrop_freq =  [ 0.5, 0.5, 0.5, 0.5, 1.0 ];

for bidx = 1:length(bandlist)

  bandoverrides_mag(bidx) = struct( 'seg', struct( ...
    'dbpeak', dbpeak_mag(bidx), 'dbend', dbend_mag(bidx), ...
    'qlong', qlong_mag(bidx), 'qdrop', qdrop_mag(bidx) ...
    ), 'param', struct() );

  bandoverrides_freq(bidx) = struct( 'seg', struct( ...
    'dbpeak', dbpeak_freq(bidx), 'dbend', dbend_freq(bidx), ...
    'qlong', qlong_freq(bidx), 'qshort', qshort_freq(bidx), ...
    'qdrop', qdrop_freq(bidx) ...
    ), 'param', struct() );

end


%
%
% Banner.

disp('== Loading Field Trip data.');


%
% Load the datafile.

isok = true;

load(inputfile, inputvar);

% Rename this to something consistent and sensible.
% NOTE - This _should_ be smart enough to copy-on-modify only, but we should
% make sure we have enough headroom for duplication just in case.
rawdata = eval(inputvar);


% Make sure we have a valid sampling rate.
```

```
ftrate = wlFT_getSamplingRate(rawdata);

if isnan(ftrate)
  isok = false;
  disp('### Couldn''t get sampling rate.');
% FIXME - Debugging.
else
  disp(sprintf('Sampling rate:  %d', round(ftrate)));
end


% FIXME - Load Thilo-specific information.

if isok && use_thilo_specific

  load(inputfile, 'tinfo');

  thilo_rewardsamps = [];

  scratch = tinfo.rewOnTimes;

  for tidx = 1:length(scratch);
    % Convert a negative timestamp in seconds into a positive sample offset.
    thilo_rewardsamps(tidx) = round( - ftrate * scratch{tidx}(1) );
  end

  clear scratch;


  % Add the stop time as an extra field in "trialinfo".

  auxidx = size(rawdata.trialinfo);
  auxidx = auxidx(2) + 1;

  rawdata.trialinfo(:,auxidx) = thilo_rewardsamps;


  % Make a crop function that checks this new endpoint.

  cropfunc = @(thistrial, sampinforow, trialinforow) ...
    deal( 1, trialinforow(auxidx) );

end


% FIXME - Not doing automatic artifact identification yet.
% We do need it; about 10% of trials are bad.
```

```
disp('-- Field Trip data loaded.');


%
% Detect events in the FT data, and make plots.

if isok

  disp(sprintf( '.. %d trials in dataset.', length(rawdata.trial) ));


  % Extract our desired channels.

  [ rawdata chandefs ] = wlFT_getChanSubset(rawdata, targetchannels);

  if 1 > length(chandefs)
    isok = false;
    disp('### Couldn''t identify target channels.');
  end

end

if isok

  % Preprocessing step: Trim the trials.

  [ trimmeddata cropdefs ] = wlFT_trimTrials(rawdata, cropfunc);


  % Preprocessing step: Apply a notch filter to remove power line noise.
  % This shows up in a fraction of the trials.
  % Do this _after_ we've trimmed the electrical artifacts, to avoid large
  % excursions. We may still get edge effects.
  % NOTE - We're using "filtfilt", which gives no phase shift but twice the
  % filter order.

  if use_notch

    disp('-- Applying notch filter.');

    powerfilt = designfilt( 'bandstopiir', 'SampleRate', ftrate, ...
      'HalfPowerFrequency1', notch_center - notch_halfwidth, ...
      'HalfPowerFrequency2', notch_center + notch_halfwidth, ...
      'FilterOrder', notch_order );

    for tidx = 1:length(trimmeddata.trial)

      thistrial = trimmeddata.trial{tidx};
      [ chancount, sampcount ] = size(thistrial);

      for cidx = 1:chancount
        thischannel = thistrial(cidx, :);
        thischannel = filtfilt(powerfilt, thischannel);
        thistrial(cidx, :) = thischannel;
      end

      trimmeddata.trial{tidx} = thistrial;

    end

    disp('-- Finished applying notch filter.');

  end


  % FIXME - Clip the band list for testing.
  if use_one_band
    bandlist = bandlist(one_band_chosen:one_band_chosen);
    bandoverrides_mag = bandoverrides_mag(one_band_chosen:one_band_chosen);
    bandoverrides_freq = bandoverrides_freq(one_band_chosen:one_band_chosen);
  end


  %
  %
  % Do event detection.

  disp('== Detecting using magnitude.');


  newmatrix_mag = wlFT_doFindEventsInTrials_MT(trimmeddata, bandlist, ...
    segconfig_mag, paramconfig_chosen, bandoverrides_mag, true );
  newmatrix_mag = wlFT_pruneEventsByTime(newmatrix_mag, ...
    padtimestart, padtimeend);

  newdata_mag = wlFT_getEventTrialsFromMatrix(newmatrix_mag);


  disp('== Detecting using frequency stability.');


  newmatrix_freq = wlFT_doFindEventsInTrials_MT(trimmeddata, bandlist, ...
    segconfig_freq, paramconfig_chosen, bandoverrides_freq, true );
  newmatrix_freq = wlFT_pruneEventsByTime(newmatrix_freq, ...
    padtimestart, padtimeend);

  newdata_freq = wlFT_getEventTrialsFromMatrix(newmatrix_freq);
```

```
disp('== Finished detecting.');


%
%
% Do plotting.

disp('-- Plotting detected events.');
disp(datetime);


% Plot by matrix.

wlPlot_plotAllMatrixEvents(figconfig, newmatrix_mag, ...
  'Mag Recon', 'mag', bandplots_max, stride_trial, stride_channel);
wlPlot_plotAllMatrixEvents(figconfig, newmatrix_freq, ...
  'Freq Recon', 'freq', bandplots_max, stride_trial, stride_channel);


disp(datetime);
disp('-- Finished plotting events.');


%
%
% Collect and report event statistics.


[ bandcount trialcount chancount ] = size(newmatrix_mag.events);

for bidx = 1:bandcount

  thiscountmag = 0;
  thiscountfreq = 0;

  for tidx = 1:trialcount
    for cidx = 1:chancount

      evlistmag = newmatrix_mag.events{bidx, tidx, cidx};
      evlistfreq = newmatrix_freq.events{bidx, tidx, cidx};

      thiscountmag = thiscountmag + length(evlistmag);
      thiscountfreq = thiscountfreq + length(evlistfreq);
    end
  end

  disp(sprintf( '%s band:   (Mag)  %d events  (%.1f per trial)', ...
    bandlist(bidx).name, ...
    thiscountmag, thiscountmag / (trialcount * chancount) ));
    disp(sprintf( '%s band:   (Freq) %d events  (%.1f per trial)', ...
      bandlist(bidx).name, ...
      thiscountfreq, thiscountfreq / (trialcount * chancount) ));
  end


  %
  %
  % Save detected events.

  disp('-- Saving events to disk.');

% FIXME - Make a copy of the trimmed-coordinate events, for debugging.
% This also includes channel pruning.
trimmed_mag = newdata_mag;
trimmed_freq = newdata_freq;

  % Align sample indices back to un-trimmed data.
  newdata_mag = wlFT_unTrimMetadata(newdata_mag, cropdefs);
  newdata_freq = wlFT_unTrimMetadata(newdata_freq, cropdefs);

  % Map channels back to the original channel indices.
  newdata_mag = wlFT_unMapChannels(newdata_mag, chandefs);
  newdata_freq = wlFT_unMapChannels(newdata_freq, chandefs);

  save( outputfile, ...
    'newmatrix_mag', 'newmatrix_freq', ...
    'cropdefs', 'chandefs', ...
    'newdata_mag', 'newdata_freq' );

  disp('-- Finished saving events.');

end  % "isok" check.


%
%
% Banner.

disp('== Finished detecting events in Field Trip data.');



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

Appendix D

Sample Code – Synthetic Data

D.1 do_ft_synth.m

```
% Field Trip-related test scripts - Synthetic FT data.
%
% See LICENSE.md for distribution information.

%
%
% Includes.

do_ft_init



%
%
% Configuration.

% NOTE - This is moved to its own file, so that we can invoke it elsewhere.

do_ft_synth_config



%
%
% Generation.


%
% Make a synthetic trace, if we don't already have one.

if ~exist('traceft', 'var')
```

```
  do_ft_synth_gen
end



%
%
% Event detection.


% Do event detection, if we haven't already.

if ~exist('detectft_mag', 'var')
  do_ft_synth_detect
end



%
%
% Event post-processing.

do_ft_synth_post



%
%
% Get event results for the "chosen" threshold values.
% Get confusion matrix counts and lists.


do_ft_synth_calc



%
%
% Plotting.


do_ft_synth_plot



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

D.2 do_ft_synth_gen.m

```
% Field Trip-related test scripts - Synthetic FT data - Data generation.

%
% Includes.

% FIXME - Assume init has already been called.


%
%
% Configuration.


% Burst type definitions.
% These are band-agnostic.

% Short pulses, not chirped or ramped.
burstpeep.trange = [ 1.0 2.0 ];
burstpeep.framp = [ 1.0 1.0 ];
burstpeep.aramp = [ 1.0 1.0 ];

% Medium length pulses with strong chirping and ramping.
burstchirp.trange = [ 2.0 4.0 ];
burstchirp.framp = [ 0.67 1.5 ];
burstchirp.aramp = [ 0.33 3.0 ];

% Long pulses with weak chirping and ramping. "Fish in a barrel" case.
bursttone.trange = [ 3.0 6.0 ];
bursttone.framp = [ 0.8 1.25 ];
bursttone.aramp = [ 0.75 1.33 ];


% Burst occurrence parameters.

% Synthetic burst traces with realistic content.
% Maybe a bit more frequent than usual.

clear burstdefs;
clear thisdef;

% Allow lots of quiet events.
thisdef.snrrange = [ -20 20 ]; % dB; amplitude 0.1x-10x.

% Theta band: peeps (max duration about 0.5 sec).

thisdef.rate = 0.5;
thisdef.noiseband = bandtheta;
thisdef.fctrrange = bandtheta;
thisdef.durrange = burstpeep.trange;
thisdef.framprange = burstpeep.framp;
thisdef.aramprange = burstpeep.aramp;

burstdefs(1) = thisdef;

% Alpha band: tones and chirps (max duration about 1 sec tones, 0.5 chirps).

thisdef.rate = 0.2;
thisdef.noiseband = bandalpha;
thisdef.fctrrange = bandalpha;
thisdef.durrange = bursttone.trange;
thisdef.framprange = bursttone.framp;
thisdef.aramprange = bursttone.aramp;

burstdefs(2) = thisdef;

thisdef.rate = 0.4;
thisdef.noiseband = bandalpha;
thisdef.fctrrange = bandalpha;
thisdef.durrange = burstchirp.trange;
thisdef.framprange = burstchirp.framp;
thisdef.aramprange = burstchirp.aramp;

burstdefs(3) = thisdef;

% Beta band: chirps (max duration 1/3 sec).

thisdef.rate = 1;
thisdef.noiseband = bandbeta;
thisdef.fctrrange = bandbeta;
thisdef.durrange = burstchirp.trange;
thisdef.framprange = burstchirp.framp;
thisdef.aramprange = burstchirp.aramp;

burstdefs(4) = thisdef;

% Gamma band: chirps (max duration 1/8 sec).

thisdef.rate = 2;
thisdef.noiseband = bandgamma;
thisdef.fctrrange = bandgamma;
thisdef.durrange = burstchirp.trange;
thisdef.framprange = burstchirp.framp;
thisdef.aramprange = burstchirp.aramp;

burstdefs(5) = thisdef;
```

```
%
%
% Generate waveform data.


disp('-- Generating synthetic waveforms.');
disp(datetime);

[ traceft groundft ] = wlSynth_genFieldTrip(fsamp, synth_chancount, ...
  synth_trialcount, synth_trialdur, burstdefs, synth_channelratevar, ...
  synth_channelnoisevar);


disp('-- Splitting waveforms by band.');
disp(datetime);


groundftbyband = wlAux_splitEvMatrixByBand(groundft, bandlist);


disp('-- Saving synthetic waveforms to disk.');
disp(datetime);


save( synth_fname_data, 'traceft', 'groundft', 'groundftbyband' );


disp('-- Finished generating waveforms.');


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

D.3 do_ft_synth_detect.m

```
% Field Trip-related test scripts - Synthetic FT data - Event detection.

%
%
% Includes.

% FIXME - Assume init has already been called.



%
%
% Configuration.
% Select the actual parameter extraction method to use.

paramconfig_chosen = paramconfig_coarse;

if strcmp('amp', param_chosen_type)

  disp('.. Using detection type "amp".');
  paramconfig_chosen = paramconfig_amp;

elseif strcmp('both', param_chosen_type)

  disp('.. Using detection type "both".');
  paramconfig_chosen = paramconfig_both;

else

  disp('.. Using detection type "coarse".');

end


% Band-specific parameter overrides.

for bidx = 1:length(bandlist)

  bandoverrides_mag(bidx) = struct( ...
    'seg', struct( 'dbpeak', magthreshdb(bidx) ), ...
    'param', struct() );

  bandoverrides_freq(bidx) = struct( ...
    'seg', struct( 'dbpeak', freqthreshdb(bidx) ), ...
    'param', struct() );

end


% Use DC average instead of "DC" LPF for the specified bands.

for bidx = 1:length(bandlist)
  if seg_use_dc(bidx)
    bandoverrides_mag(bidx).seg.qlong_freq = inf;
    bandoverrides_freq(bidx).seg.qlong_freq = inf;
  end
end



%
```

```
%
% Event detection.


disp('== Detecting using magnitude.');
disp(datetime);

clear detectft_mag;

for thidx = 1:length(detsweepthresholds)

  disp(sprintf( '-- Threshold:  %d dB', round(detsweepthresholds(thidx)) ));

  % Make a temporary copy of the "band overrides" structure, so that the
  % original stays intact.
  bandoverrides_test = bandoverrides_mag;
  for bidx = 1:length(bandlist)
    bandoverrides_test(bidx).seg.dbpeak = detsweepthresholds(thidx);
  end

  detect_temp = wlFT_doFindEventsInTrials_MT(traceft, bandlist, ...
    segconfig_mag, paramconfig_chosen, bandoverrides_test, true );

  % FIXME - Prune this, to avoid roll-off window issues.
  detect_temp = wlFT_pruneEventsByTime(detect_temp, padtime, padtime);

  % Save the result.
  detectft_mag{thidx} = detect_temp;

end


disp('== Detecting using frequency stability.');
disp(datetime);

clear detectft_freq;

for thidx = 1:length(detsweepthresholds)

  disp(sprintf( '-- Threshold:  %d dB', round(detsweepthresholds(thidx)) ));

  % Make a temporary copy of the "band overrides" structure, so that the
  % original stays intact.
  bandoverrides_test = bandoverrides_freq;
  for bidx = 1:length(bandlist)
    bandoverrides_test(bidx).seg.dbpeak = detsweepthresholds(thidx);
  end

  detect_temp = wlFT_doFindEventsInTrials_MT(traceft, bandlist, ...
    segconfig_freq, paramconfig_chosen, bandoverrides_test, true );
  % FIXME - Prune this, to avoid roll-off window issues.
  detect_temp = wlFT_pruneEventsByTime(detect_temp, padtime, padtime);

  detectft_freq{thidx} = detect_temp;

end


disp(datetime);
disp('== Finished detecting.');



%
% Save the results.

disp('-- Saving detected events to disk.')

% Main saved data is the array of detection matrices indexed by threshold.

% Also save the detection configuration information so that other scripts can
% use it without having to cut-and-paste or re-generate it.

% FIXME - Need to use "-v7.3", as decent-sized data gives detection
% structures that are > 2 GB.
% NOTE - '-nocompression' is faster but considerably larger.
% Use "whos detectft_mag detectft_freq" to see how large.

save( synth_fname_detect, ...
  'segconfig_mag', 'segconfig_freq', 'paramconfig_chosen', ...
  'bandoverrides_mag', 'bandoverrides_freq', ...
  'detectft_mag', 'detectft_freq', ...
  '-v7.3' );

disp('-- Finished saving detected events.')


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

D.4 do_ft_synth_post.m

```
% Field Trip-related test scripts - Synthetic FT data - Post-processing

%
%
```

```
% Includes.

% FIXME - Assume init has already been called.



%
%
% Configuration.


% Threshold decimation factor for plotting.
threshdecim = 2;


% Wave comparison functions.

wavecomparefuncwide = @(thisev, thiswave) ...
  wlProc_calcWaveErrorRelative( thiswave.ftwave, thisev.sampstart, ...
    thisev.wave, thisev.s1 );

wavecomparefuncbpf = @(thisev, thiswave) ...
  wlProc_calcWaveErrorRelative( thiswave.bpfwave, thisev.sampstart, ...
    thisev.wave, thisev.s1 );



%
%
% Compute relative RMS error (vs the BPF signal and WB signal).


groundftbyband = ...
  wlFT_calcEventErrors(groundftbyband, wavecomparefuncbpf, 'errbpf');
groundftbyband = ...
  wlFT_calcEventErrors(groundftbyband, wavecomparefuncwide, 'errwide');

threshcount = length(detectft_mag);
for thidx = 1:threshcount
  detectft_mag{thidx} = wlFT_calcEventErrors( ...
    detectft_mag{thidx}, wavecomparefuncbpf, 'errbpf' );
  detectft_mag{thidx} = wlFT_calcEventErrors( ...
    detectft_mag{thidx}, wavecomparefuncwide, 'errwide' );
end

threshcount = length(detectft_freq);
for thidx = 1:threshcount
  detectft_freq{thidx} = wlFT_calcEventErrors( ...
    detectft_freq{thidx}, wavecomparefuncbpf, 'errbpf' );
  detectft_freq{thidx} = wlFT_calcEventErrors( ...
    detectft_freq{thidx}, wavecomparefuncwide, 'errwide' );
end




%
%
% Plotting.

% FIXME - This should really be in do_ft_synth_plot.m, but keep it here to
% avoid having to save copies of the un-pruned event matrices.


if plot_ftsyn_raw_error

  wlPlot_plotMatrixErrorStats( figconfig, ...
    [ struct( 'evmatrix', groundftbyband, 'errfield', 'errwide', ...
      'legend', 'wideband', 'color', cbrn ), ...
      struct( 'evmatrix', groundftbyband, 'errfield', 'errbpf', ...
      'legend', 'bandpass', 'color', cblu ) ], ...
    'Synthetic Ground Truth Error', 'gt' );

  % FIXME - We want to plot this for "selected" thresholds, not all of them.
  % Right now selection happens after pruning, not before. Decimate instead.

  for thidx = 1:threshdecim:length(detsweepthresholds)
    thisdb = detsweepthresholds(thidx);

    wlPlot_plotMatrixErrorStats( figconfig, ...
      [ struct( 'evmatrix', detectft_mag{thidx}, 'errfield', 'errwide', ...
        'legend', 'mag wideband', 'color', ccyn ), ...
        struct( 'evmatrix', detectft_freq{thidx}, 'errfield', 'errwide', ...
        'legend', 'freq wideband', 'color', cyel ), ...
        struct( 'evmatrix', detectft_mag{thidx}, 'errfield', 'errbpf', ...
        'legend', 'mag bandpass', 'color', cblu ), ...
        struct( 'evmatrix', detectft_freq{thidx}, 'errfield', 'errbpf', ...
        'legend', 'freq bandpass', 'color', cbrn ) ], ...
      sprintf('Synthetic Detection Error (%d dB)', thisdb), ...
      sprintf('det-%02d', thisdb) );
  end

end




%
%
% Prune regions of parameter/error space with lots of false positives.
% FIXME - Throw out anything with BPF RMS error over 0.7.
% This is about the best we can do without losing many true positives.
% In combination with well-tuned thresholds, this does help.

passfunc = @(thisev) (0.7 >= thisev.auxdata.errbpf);

threshcount = length(detectft_mag);
for thidx = 1:threshcount
  detectft_mag{thidx} = wlAux_pruneMatrix(detectft_mag{thidx}, passfunc);
end

threshcount = length(detectft_freq);
for thidx = 1:threshcount
  detectft_freq{thidx} = wlAux_pruneMatrix(detectft_freq{thidx}, passfunc);
end



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

D.5 do_ft_synth_calc.m

```
% Field Trip-related test scripts - Synthetic FT data - Calculation

%
%
% Includes.

% FIXME - Assume init has already been called.



%
%
% Get event results for the "chosen" threshold values.


disp('-- Selecting event lists with chosen thresholds.');


detectft_mag_selected = detectft_mag{1};
detectft_freq_selected = detectft_freq{1};


%
% Selected thresholds.
```

```
for bidx = 1:length(bandlist)

  thidx = do_ft_calcFindThresholdIndex(magthreshdb(bidx), detsweepthresholds);

  detectft_mag_selected.events(bidx,:,:) = ...
    detectft_mag{thidx}.events(bidx,:,:);


  thidx = do_ft_calcFindThresholdIndex(freqthreshdb(bidx), detsweepthresholds);

  detectft_freq_selected.events(bidx,:,:) = ...
    detectft_freq{thidx}.events(bidx,:,:);

end


%
% Low/medium/high thresholds near the selected thresholds.

for swidx = 1:length(plotthreshdbmag)

  detectft_mag_swept(swidx) = detectft_mag_selected;

  for bidx = 1:length(bandlist)

    thidx = do_ft_calcFindThresholdIndex( ...
      magthreshdb(bidx) + plotthreshdbmag(swidx), detsweepthresholds);
    detectft_mag_swept(swidx).events(bidx,:,:) = ...
      detectft_mag{thidx}.events(bidx,:,:);

  end

end

for swidx = 1:length(plotthreshdbfreq)

  detectft_freq_swept(swidx) = detectft_freq_selected;

  for bidx = 1:length(bandlist)

    thidx = do_ft_calcFindThresholdIndex( ...
      freqthreshdb(bidx) + plotthreshdbfreq(swidx), detsweepthresholds);
    detectft_freq_swept(swidx).events(bidx,:,:) = ...
      detectft_freq{thidx}.events(bidx,:,:);

  end

end
```

```
%
% Collapsed event lists, merging across all axes.
% FIXME - This can take O(n^2) time!

% NOTE - This produces overlapping events. The result is only useful for
% statistics and plotting; comparing it against ground truth will lose most
% of the events (they'll be flagged as false positives).


% NOTE - Collapsing ground truth, too.
% Starting with the wideband version, so already one band.
groundft_merged = wlAux_mergeTrialsChannels(groundft, evcomparefuncfalse);

detectft_mag_mergedselect = wlAux_splitEvMatrixByBand( ...
  wlAux_mergeTrialsChannels(detectft_mag_selected, evcomparefuncfalse), ...
  bandlistwide );

for swidx = 1:length(detectft_mag_swept)
  detectft_mag_mergedswept(swidx) = wlAux_splitEvMatrixByBand( ...
    wlAux_mergeTrialsChannels( ...
      detectft_mag_swept(swidx), evcomparefuncfalse ), ...
    bandlistwide );
end

detectft_freq_mergedselect = wlAux_splitEvMatrixByBand( ...
  wlAux_mergeTrialsChannels(detectft_freq_selected, evcomparefuncfalse), ...
  bandlistwide );

for swidx = 1:length(detectft_freq_swept)
  detectft_freq_mergedswept(swidx) = wlAux_splitEvMatrixByBand( ...
    wlAux_mergeTrialsChannels( ...
      detectft_freq_swept(swidx), evcomparefuncfalse ), ...
    bandlistwide );
end


disp('-- Finished selecting events.');




%
%
% Get confusion matrix counts and lists.

% This is fast to compute, and quite large. Matlab doesn't want to save it.
% Saving can be forced by using version 7.3 and up, but there's no need to.


disp('-- Computing confusion matrix statistics.')
disp(datetime);


%
% Per-threshold statistics, across channels and trials and bands.

clear ftgtstats_mag;
clear ftgtstats_freq;

for thidx = 1:length(detsweepthresholds)

  [ fpmat fnmat tpmat matchtruth matchtest missingtruth missingtest ] = ...
    wlFT_compareMatrixEventsVsTruth( ...
      groundftbyband, detectft_mag{thidx}, evcomparefunc );
  ftgtstats_mag{thidx} = struct( ...
    'fp', fpmat, 'fn', fnmat, 'tp', tpmat, ...
    'truematch', matchtruth, 'truemissing', missingtruth, ...
    'testmatch', matchtest, 'testmissing', missingtest );

  [ fpmat fnmat tpmat matchtruth matchtest missingtruth missingtest ] = ...
    wlFT_compareMatrixEventsVsTruth( ...
      groundftbyband, detectft_freq{thidx}, evcomparefunc );
  ftgtstats_freq{thidx} = struct( ...
    'fp', fpmat, 'fn', fnmat, 'tp', tpmat, ...
    'truematch', matchtruth, 'truemissing', missingtruth, ...
    'testmatch', matchtest, 'testmissing', missingtest );

end


%
% Statistics with selected thresholds, across channels and trials and bands.


% Single "chosen" threshold.

[ fpmat fnmat tpmat matchtruth matchtest missingtruth missingtest ] = ...
  wlFT_compareMatrixEventsVsTruth( ...
    groundftbyband, detectft_mag_selected, evcomparefunc );
ftgtstats_mag_selected = struct( ...
  'fp', fpmat, 'fn', fnmat, 'tp', tpmat, ...
  'truematch', matchtruth, 'truemissing', missingtruth, ...
  'testmatch', matchtest, 'testmissing', missingtest );

[ fpmat fnmat tpmat matchtruth matchtest missingtruth missingtest ] = ...
  wlFT_compareMatrixEventsVsTruth( ...
    groundftbyband, detectft_freq_selected, evcomparefunc );
ftgtstats_freq_selected = struct( ...
  'fp', fpmat, 'fn', fnmat, 'tp', tpmat, ...
  'truematch', matchtruth, 'truemissing', missingtruth, ...
  'testmatch', matchtest, 'testmissing', missingtest );


% "Low"/"middle"/"high" thresholds.

clear ftgtstats_mag_swept;
clear ftgtstats_freq_swept;

for swidx = 1:length(plotthreshdbmag)

  [ fpmat fnmat tpmat matchtruth matchtest missingtruth missingtest ] = ...
    wlFT_compareMatrixEventsVsTruth( ...
      groundftbyband, detectft_mag_swept(swidx), evcomparefunc );
  ftgtstats_mag_swept(swidx) = struct( ...
    'fp', fpmat, 'fn', fnmat, 'tp', tpmat, ...
    'truematch', matchtruth, 'truemissing', missingtruth, ...
    'testmatch', matchtest, 'testmissing', missingtest );

end

for swidx = 1:length(plotthreshdbfreq)

  [ fpmat fnmat tpmat matchtruth matchtest missingtruth missingtest ] = ...
    wlFT_compareMatrixEventsVsTruth( ...
      groundftbyband, detectft_freq_swept(swidx), evcomparefunc );
  ftgtstats_freq_swept(swidx) = struct( ...
    'fp', fpmat, 'fn', fnmat, 'tp', tpmat, ...
    'truematch', matchtruth, 'truemissing', missingtruth, ...
    'testmatch', matchtest, 'testmissing', missingtest );

end


%
% Statistics with selected thresholds, merged into one trial/channel/band.

% NOTE - This produces overlapping events. The result is only useful for
% statistics and plotting; comparing it against ground truth will lose most
% of the events (they'll be flagged as false positives).


% Single "chosen" threshold.

ftgtstats_mag_mergedselect = wlHelper_collapseStats( ...
  ftgtstats_mag_selected, bandwide, evcomparefuncfalse );

ftgtstats_freq_mergedselect = wlHelper_collapseStats( ...
  ftgtstats_freq_selected, bandwide, evcomparefuncfalse );
```

```
% "Low"/"middle"/"high" thresholds.

clear ftgtstats_mag_mergedswept;
clear ftgtstats_freq_mergedswept;

for swidx = 1:length(plotthreshdbmag)
  ftgtstats_mag_mergedswept(swidx) = wlHelper_collapseStats( ...
    ftgtstats_mag_swept(swidx), bandwide, evcomparefuncfalse );
end

for swidx = 1:length(plotthreshdbfreq)
  ftgtstats_freq_mergedswept(swidx) = wlHelper_collapseStats( ...
    ftgtstats_freq_swept(swidx), bandwide, evcomparefuncfalse );
end


%
% Done.

disp(datetime);
disp('-- Finished computing statistics.')



%
%
% Helper functions.


% This function collapses confusion matrix statistics and lists across bands,
% trials, and channels.
%
% NOTE - This produces overlapping events. The result is only useful for
% statistics and plotting; comparing it against ground truth will lose most
% of the events (they'll be flagged as false positives).
%
% "oldstats" is a statistics structure of the type used above.
% "banddef" [ min max ] defines the single output band (usually wideband).
% "comparefunc" is a function used to determine when events should be merged
%   (i.e. duplicates). Form is per COMPAREFUNC.txt.

function newstats = wlHelper_collapseStats(oldstats, banddef, comparefunc)

  matchtruth =   oldstats.truematch;
  missingtruth = oldstats.truemissing;
  matchtest =    oldstats.testmatch;
  missingtest =  oldstats.testmissing;

  bandlist = [ struct( 'band', banddef, 'label', 'all', 'name', 'Merged' ) ];
```

```
% Before doing the merge: annotate the "match" lists with position
% information.

truthcount = 0;
testcount = 0;

% Size is the same for both lists.
[ bandcount trialcount chancount ] = size(matchtruth.events);

for bidx = 1:bandcount
  for tidx = 1:trialcount
    for cidx = 1:chancount

      thislist = matchtruth.events{bidx, tidx, cidx};
      for eidx = 1:length(thislist)
        truthcount = truthcount + 1;
        thislist(eidx).auxdata.matchidx = truthcount;
      end
      matchtruth.events{bidx, tidx, cidx} = thislist;

      thislist = matchtest.events{bidx, tidx, cidx};
      for eidx = 1:length(thislist)
        testcount = testcount + 1;
        thislist(eidx).auxdata.matchidx = testcount;
      end
      matchtest.events{bidx, tidx, cidx} = thislist;

    end
  end
end


% Merge lists.

matchtruth = wlAux_splitEvMatrixByBand( ...
  wlAux_mergeTrialsChannels(matchtruth, comparefunc), bandlist );
missingtruth = wlAux_splitEvMatrixByBand( ...
  wlAux_mergeTrialsChannels(missingtruth, comparefunc), bandlist );
matchtest = wlAux_splitEvMatrixByBand( ...
  wlAux_mergeTrialsChannels(matchtest, comparefunc), bandlist );
missingtest = wlAux_splitEvMatrixByBand( ...
  wlAux_mergeTrialsChannels(missingtest, comparefunc), bandlist );


% Put the match lists in consistent order.
% FIXME - They're no longer sorted by starting sample time this way.

thislist = matchtruth.events{1,1,1};
if 0 < length(thislist)
```

```
    idxlist = [];
    for eidx = 1:length(thislist)
      idxlist(eidx) = thislist(eidx).auxdata.matchidx;
    end
    [ sortedidx sortedidxlut ] = sort(idxlist);
    matchtruth.events{1,1,1} = thislist(sortedidxlut);

  end

  thislist = matchtest.events{1,1,1};
  if 0 < length(thislist)

    idxlist = [];
    for eidx = 1:length(thislist)
      idxlist(eidx) = thislist(eidx).auxdata.matchidx;
    end
    [ sortedidx sortedidxlut ] = sort(idxlist);
    matchtest.events{1,1,1} = thislist(sortedidxlut);

  end


  % Build the new statistics structure.

  newstats = struct( ...
    'fp', length(missingtest.events{1,1,1}), ...
    'fn', length(missingtruth.events{1,1,1}), ...
    'tp', ...
      min( length(matchtruth.events{1,1,1}), ...
        length(matchtest.events{1,1,1}) ), ...
    'truematch', matchtruth, 'truemissing', missingtruth, ...
    'testmatch', matchtest, 'testmissing', missingtest );

% Done.

end


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

D.6 do_ft_synth_plot.m

```
% Field Trip-related test scripts - Synthetic FT data - Plotting

%
```

```
%
% Includes.

% FIXME - Assume init has already been called.



%
%
% Configuration.


% Scatter-plot ranges.

plotband = [ 0.5 * bandtheta(1) 1.5*bandgamma(2) ];
plotdur = [ 0 7 ];
plotamp = [ 1e-2 10 ];

% FIXME - Expand ranges, in case of plotting oddities.
if true
  plotdur = [ 0 15 ];
  plotamp = [ 1e-2 100 ];
end



%
%
% Plot ground truth events.


if plot_ftsyn_gt_band || plot_ftsyn_gt_wide
  disp('-- Plotting ground truth events.');
  disp(datetime);
end


% Per-band.

if plot_ftsyn_gt_band
  wlPlot_plotAllMatrixEvents(figconfig, groundftbyband, ...
    'Synthetic Ground Truth', 'ftgtbb', ...
    bandplots_max, stride_trial, stride_channel);
end

% Wideband.

if plot_ftsyn_gt_wide
  % FIXME - Manually tweak spectrum display.
  groundft.bandinfo(1).band = [ 1 50 ];
  wlPlot_plotAllMatrixEvents(figconfig, groundft, ...
    'Synthetic Ground Truth', 'ftgt', ...
    bandplots_max, stride_trial, stride_channel);
end



%
%
% Plot detected events.


if plot_ftsyn_det_waves

  disp('-- Plotting detected event waveforms.');
  disp(datetime);

  wlPlot_plotAllMatrixEvents(figconfig, detectft_mag_selected, ...
    'Synthetic Mag Detect', 'ftmag', ...
    bandplots_max, stride_trial, stride_channel);

  wlPlot_plotAllMatrixEvents(figconfig, detectft_freq_selected, ...
    'Synthetic Freq Detect', 'ftfreq', ...
    bandplots_max, stride_trial, stride_channel);

end



%
%
% Plot detection rates vs threshold.


if plot_ftsyn_det_rates

  disp('-- Plotting detection rate curves.');
  disp(datetime);

  for bidx = 1:length(bandlist)

    % Initialize.

    bandname = bandlist(bidx).name;
    bandlabel = bandlist(bidx).label;

    chartmagstats = struct( 'fp', [], 'fn', [], 'tp', [] );
    chartfreqstats = struct( 'fp', [], 'fn', [], 'tp', [] );
    % Compile statistics.

    for thidx = 1:length(detsweepthresholds)

      thisconf = ftgtstats_mag{thidx};

      chartmagstats.fp(thidx) = sum(sum( thisconf.fp(bidx,:,:) ));
      chartmagstats.fn(thidx) = sum(sum( thisconf.fn(bidx,:,:) ));
      chartmagstats.tp(thidx) = sum(sum( thisconf.tp(bidx,:,:) ));

      thisconf = ftgtstats_freq{thidx};

      chartfreqstats.fp(thidx) = sum(sum( thisconf.fp(bidx,:,:) ));
      chartfreqstats.fn(thidx) = sum(sum( thisconf.fn(bidx,:,:) ));
      chartfreqstats.tp(thidx) = sum(sum( thisconf.tp(bidx,:,:) ));

    end


    % Plot detection curves.

    wlPlot_plotDetectCurves( figconfig, ...
      detsweepthresholds, 'Detection Threshold (dB)', ...
      [ struct( 'fp', chartmagstats.fp, 'fn', chartmagstats.fn, ...
          'tp', chartmagstats.tp, ...
          'color', cblu, 'label', 'magnitude detect' ), ...
        struct( 'fp', chartfreqstats.fp, 'fn', chartfreqstats.fn, ...
          'tp', chartfreqstats.tp, ...
          'color', cbrn, 'label', 'frequency detect' ) ], ...
      sprintf('Magnitude- and Frequency-Based Detection - %s', bandname), ...
      sprintf('all-%s', bandlabel) );

  end  % Band iteration.

end



%
%
% Scatter-plot extracted and ground truth parameters.

if plot_ftsyn_det_scatter

  disp('-- Plotting detected vs ground truth parameters.');
  disp(datetime);

  %
  % Ground truth parameters.
```

```
wlPlot_plotEventScatterMulti(figconfig, ...
  [ struct( 'eventlist', groundft_merged.events{1}, 'color', cgrn, ...
      'legend', 'ground truth' ) ], ...
  plotband, plotdur, plotamp, 'Synthetic FT - Ground Truth', 'gtruth' );


%
% Extracted parameters.


clear scatterlist;

for swidx = 1:length(plotthreshdbmag)
  scatterlist(swidx) = struct( ...
    'eventlist', detectft_mag_mergedswept(swidx).events{1}, ...
    'color', plotthreshcolors{swidx}, 'legend', ...
    sprintf('%s threshold', plotthreshnames{swidx}) );
end

wlPlot_plotEventScatterMulti( figconfig, scatterlist, ...
  plotband, plotdur, plotamp, ...
  'Synthetic FT - Magnitude Threshold Detection', 'mag-multi' );


clear scatterlist;

for swidx = 1:length(plotthreshdbfreq)
  scatterlist(swidx) = struct( ...
    'eventlist', detectft_freq_mergedswept(swidx).events{1}, ...
    'color', plotthreshcolors{swidx}, 'legend', ...
    sprintf('%s threshold', plotthreshnames{swidx}) );
end

wlPlot_plotEventScatterMulti( figconfig, scatterlist, ...
  plotband, plotdur, plotamp, ...
  'Synthetic FT - Frequency Stability Detection', 'freq-multi' );


%
% True and false positives.

for swidx = 1:length(plotthreshdbmag)

  wlPlot_plotEventScatterMulti( figconfig, ...
    [ struct( ...
      'eventlist', ...
        ftgtstats_mag_mergedswept(swidx).testmissing.events{1}, ...
      'color', cbrn, 'legend', 'Incorrect' ), ...
      struct( ...
      'eventlist', ...
        ftgtstats_mag_mergedswept(swidx).testmatch.events{1}, ...
      'color', cblu, 'legend', 'Correct' ) ], ...
    plotband, plotdur, plotamp, ...
    sprintf('Synthetic FT - Magnitude Threshold - %s thresh', ...
      plotthreshnames{swidx}), ...
    sprintf('tpfp-mag-%s', plotthreshnames{swidx}) );

end


for swidx = 1:length(plotthreshdbfreq)

  wlPlot_plotEventScatterMulti( figconfig, ...
    [ struct( ...
      'eventlist', ...
        ftgtstats_freq_mergedswept(swidx).testmissing.events{1}, ...
      'color', cbrn, 'legend', 'Incorrect' ), ...
      struct( ...
      'eventlist', ...
        ftgtstats_freq_mergedswept(swidx).testmatch.events{1}, ...
      'color', cblu, 'legend', 'Correct' ) ], ...
    plotband, plotdur, plotamp, ...
    sprintf('Synthetic FT - Frequency Stability - %s thresh', ...
      plotthreshnames{swidx}), ...
    sprintf('tpfp-freq-%s', plotthreshnames{swidx}) );

end


%
% Ground truth vs detected parameters.

clear matchseriesmag;

for swidx = 1:length(plotthreshdbmag)
  matchseriesmag(swidx) = struct( ...
    'truthlist', ftgtstats_mag_mergedswept(swidx).truematch.events{1}, ...
    'testlist', ftgtstats_mag_mergedswept(swidx).testmatch.events{1}, ...
    'color', plotthreshcolors{swidx}, ...
    'legend', sprintf('%s threshold', plotthreshnames{swidx}) );
end

wlPlot_plotEventXYMultiDual(figconfig, matchseriesmag, ...
  plotband, plotdur, plotamp, ...
  'Magnitude Threshold Detection', 'mag-xy' );

clear matchseriesfreq;

for swidx = 1:length(plotthreshdbfreq)
```

```
    matchseriesfreq(swidx) = struct( ...
      'truthlist', ftgtstats_freq_mergedswept(swidx).truematch.events{1}, ...
      'testlist', ftgtstats_freq_mergedswept(swidx).testmatch.events{1}, ...
      'color', plotthreshcolors{swidx}, ...
      'legend', sprintf('%s threshold', plotthreshnames{swidx}) );
  end

  wlPlot_plotEventXYMultiDual(figconfig, matchseriesfreq, ...
    plotband, plotdur, plotamp, ...
    'Frequency Stability Detection', 'freq-xy' );

end



%
%
% Plot reconstruction error.

if plot_ftsyn_pruned_error

  disp('-- Plotting waveform reconstruction error.');
  disp(datetime);


  for swidx = 1:length(plotthreshdbmag)

    wlPlot_plotMatrixErrorStats( figconfig, ...
      [ struct( 'evmatrix', ftgtstats_mag_swept(swidx).testmissing, ...
        'errfield', 'errbpf', 'legend', 'Incorrect', 'color', cbrn ), ...
        struct( 'evmatrix', ftgtstats_mag_swept(swidx).testmatch, ...
        'errfield', 'errbpf', 'legend', 'Correct', 'color', cblu ) ], ...
      sprintf('Synthetic FT Recon Error - Magnitude Detect - %s thresh', ...
        plotthreshnames{swidx}), ...
      sprintf('tpfp-mag-%s', plotthreshnames{swidx}) );

  end


  for swidx = 1:length(plotthreshdbfreq)

    wlPlot_plotMatrixErrorStats( figconfig, ...
      [ struct( 'evmatrix', ftgtstats_freq_swept(swidx).testmissing, ...
        'errfield', 'errbpf', 'legend', 'Incorrect', 'color', cbrn ), ...
        struct( 'evmatrix', ftgtstats_freq_swept(swidx).testmatch, ...
        'errfield', 'errbpf', 'legend', 'Correct', 'color', cblu ) ], ...
      sprintf('Synthetic FT Recon Error - Frequency Detect - %s thresh', ...
        plotthreshnames{swidx}), ...
      sprintf('tpfp-freq-%s', plotthreshnames{swidx}) );
  end

end



%
%
% Done.

disp(datetime);
disp('-- Finished plotting.');



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

Appendix E

Sample Code – Defining Custom Functions

E.1 do_ft_custom.m

```
% Field Trip-related test scripts - Synthetic FT data - "Custom" tests.
%
% This tests the ability to specify "custom" (lambda-function) segmentation
% and parameter extraction algorithms.
%
% See LICENSE.md for distribution information.

%
%
% Includes.

do_ft_init

% Grandfather in "synthetic trace" configuration.
do_ft_synth_config



%
%
% Configuration.


% Threshold sweep range for mag/freq neighbourhood.
% 1x1 for smoke tests, 3x3 for rough tests, 5x5 for the real thing.

threshsweep_xy = -4:2:4;
%threshsweep_xy = -2:2:2;
%threshsweep_xy = 0:0;
```

```
tattledetect = false;
tattlepatch = true;


% Minimum true positive rate of interest (as a fraction of the best
% true positive rate).
tpfracmin = 0.8;


% Use DC average for theta thresholds.
seg_use_dc = [ true false false false false ];

% Padding, to ignore roll-off events.
padtime = 0.50;


% Select which plots we want to generate.

plot_ftsyn_gt_band = false;
plot_ftsyn_gt_wide = false;

plot_ftsyn_det_waves = false;
plot_ftsyn_det_rates = true;
plot_ftsyn_det_scatter = true;

plot_raw_error = false;
plot_pruned_error = true;


% Segmentation configuration.
% Segmentation by combined magnitude and frequency.

segmentfunc_magfreq = @(wavedata, samprate, bpfband, segconfig) ...
  wlHelper_getEvSegmentsUsingMagFreq(wavedata, samprate, bpfband, segconfig);

segconfig_union = struct( 'type', 'custom', ...
  'segmentfunc', segmentfunc_magfreq, ...
  'config_mag', segconfig_mag, 'config_freq', segconfig_freq, ...
  'thresh_mag', 10, 'thresh_freq', 10, ...
  'operation', 'union' );

segconfig_intersect = struct( 'type', 'custom', ...
  'segmentfunc', segmentfunc_magfreq, ...
  'config_mag', segconfig_mag, 'config_freq', segconfig_freq, ...
  'thresh_mag', 10, 'thresh_freq', 10, ...
  'operation', 'intersect' );


% Parameter extraction configuration.
% Parameter extraction by incremental fitting.

paramfunc_increment = @(events, waves, samprate, paramconfig) ...
  wlHelper_getEvParamsIncremental(events, waves, samprate, paramconfig);

% FIXME - NYI.
paramconfig_increment = struct( 'type', 'custom', ...
  'paramfunc', paramfunc_increment );



%
%
% Event detection.


if ~exist('detectft_union', 'var')
  do_ft_custom_detect
end



%
%
% Compare to ground truth and tabulate statistics.


disp('== Comparing detected events to ground truth.');
disp(datetime);

clear statft_union;
clear statft_intersect;

clear listft_union;
clear listft_intersect;

for thidxmag = 1:length(threshsweep_xy)
  for thidxfreq = 1:length(threshsweep_xy)

    [fp, fn, tp, matchtruth, matchtest, missingtruth, missingtest ] = ...
      wlFT_compareMatrixEventsVsTruth( groundftbyband, ...
        detectft_union{thidxmag, thidxfreq}, evcomparefunc );

    listft_union{thidxmag, thidxfreq} = ...
      struct('hit', matchtest, 'miss', missingtest);

    for bidx = 1:length(bandlist)

      statft_union.fp(bidx, thidxmag, thidxfreq) = ...
        sum(sum(  fp(bidx,:,:) ));
      statft_union.fn(bidx, thidxmag, thidxfreq) = ...
        sum(sum(  fn(bidx,:,:) ));

      statft_union.tp(bidx, thidxmag, thidxfreq) = ...
        sum(sum(  tp(bidx,:,:) ));

    end


    [fp, fn, tp, matchtruth, matchtest, missingtruth, missingtest ] = ...
      wlFT_compareMatrixEventsVsTruth( groundftbyband, ...
        detectft_intersect{thidxmag, thidxfreq}, evcomparefunc );

    listft_intersect{thidxmag, thidxfreq} = ...
      struct('hit', matchtest, 'miss', missingtest);

    for bidx = 1:length(bandlist)

      statft_intersect.fp(bidx, thidxmag, thidxfreq) = ...
        sum(sum(  fp(bidx,:,:) ));

      statft_intersect.fn(bidx, thidxmag, thidxfreq) = ...
        sum(sum(  fn(bidx,:,:) ));

      statft_intersect.tp(bidx, thidxmag, thidxfreq) = ...
        sum(sum(  tp(bidx,:,:) ));

    end

  end
end


% Get tp and fp rates.

statft_union.tprate = statft_union.tp ./ (statft_union.tp + statft_union.fp);
statft_union.fprate = statft_union.fp ./ (statft_union.tp + statft_union.fp);

statft_intersect.tprate = ...
  statft_intersect.tp ./ (statft_intersect.tp + statft_intersect.fp);
statft_intersect.fprate = ...
  statft_intersect.fp ./ (statft_intersect.tp + statft_intersect.fp);


% Find the best true positive counts within false positive rate limits.

for bidx = 1:length(bandlist)

  bestmagthresh_union{bidx} = [];
  bestfreqthresh_union{bidx} = [];
  bestmagthresh_intersect{bidx} = [];
  bestfreqthresh_intersect{bidx} = [];

  tuplecount_union = 0;
  tuplecount_intersect = 0;

  tpbest_union = 0;
  tpbest_intersect = 0;

  % First pass: Find the best tp count with acceptable fp rate.

  for thidxmag = 1:length(threshsweep_xy)
    for thidxfreq = 1:length(threshsweep_xy)

      tpcount = statft_union.tp(bidx, thidxmag, thidxfreq);
      fprate = statft_union.fprate(bidx, thidxmag, thidxfreq);
      if (tpcount > tpbest_union) && (fprate <= eventerrmax)
        tpbest_union = tpcount;
      end

      tpcount = statft_intersect.tp(bidx, thidxmag, thidxfreq);
      fprate = statft_intersect.fprate(bidx, thidxmag, thidxfreq);
      if (tpcount > tpbest_intersect) && (fprate <= eventerrmax)
        tpbest_intersect = tpcount;
      end

    end
  end
% FIXME - Diagnostics.
disp(sprintf('.. Best %s:  (U) %d tp   (I) %d tp', bandlist(bidx).name, ...
tpbest_union, tpbest_intersect));

  % Second pass: Find all rates close to this.

  for thidxmag = 1:length(threshsweep_xy)
    for thidxfreq = 1:length(threshsweep_xy)

      tpcount = statft_union.tp(bidx, thidxmag, thidxfreq);
      fprate = statft_union.fprate(bidx, thidxmag, thidxfreq);

      thismag = threshsweep_xy(thidxmag) + magthreshdb(bidx);
      thisfreq = threshsweep_xy(thidxfreq) + freqthreshdb(bidx);

      if (tpcount > 0) && (fprate <= eventerrmax) ...
        && (tpcount >= tpbest_union * tpfracmin)
        tuplecount_union = tuplecount_union + 1;
        bestmagthresh_union{bidx}(tuplecount_union) = thismag;
        bestfreqthresh_union{bidx}(tuplecount_union) = thisfreq;
% FIXME - Diagnostics.
```

```
disp(sprintf('(U) %s - M %d F %d - tp %d fpr %.2f', bandlist(bidx).name, ...
thismag, thisfreq, tpcount, fprate));
      end

      tpcount = statft_intersect.tp(bidx, thidxmag, thidxfreq);
      fprate = statft_intersect.fprate(bidx, thidxmag, thidxfreq);

      if (tpcount > 0) && (fprate <= eventerrmax) ...
        && (tpcount >= tpbest_intersect * tpfracmin)
        tuplecount_intersect = tuplecount_intersect + 1;
        bestmagthresh_intersect{bidx}(tuplecount_intersect) = thismag;
        bestfreqthresh_intersect{bidx}(tuplecount_intersect) = thisfreq;
% FIXME - Diagnostics.
disp(sprintf('(I) %s - M %d F %d - tp %d fpr %.2f', bandlist(bidx).name, ...
thismag, thisfreq, tpcount, fprate));
      end

    end
  end

end


disp(datetime);
disp('== Finished comparing to ground truth.');



%
%
% Plotting.


disp('== Plotting.');
disp(datetime);


% Contour plots.

for bidx = 1:length(bandlist)

  clear tpcounts;
  clear fpcounts;

  tpcounts(:,:) = statft_union.tp(bidx,:,:);
  fpcounts(:,:) = statft_union.fp(bidx,:,:);

  wlHelper_plotContours( figconfig, tpcounts, fpcounts, ...
    threshsweep_xy + magthreshdb(bidx), ...
    threshsweep_xy + freqthreshdb(bidx), ...
    bestmagthresh_union{bidx}, bestfreqthresh_union{bidx}, ...
    sprintf( 'Mag union Freq - %s band', bandlist(bidx).name ), ...
    sprintf( 'combo-union-%s', bandlist(bidx).label ) );

  tpcounts(:,:) = statft_intersect.tp(bidx,:,:);
  fpcounts(:,:) = statft_intersect.fp(bidx,:,:);
  wlHelper_plotContours( figconfig, tpcounts, fpcounts, ...
    threshsweep_xy + magthreshdb(bidx), ...
    threshsweep_xy + freqthreshdb(bidx), ...
    bestmagthresh_intersect{bidx}, bestfreqthresh_intersect{bidx}, ...
    sprintf( 'Mag intersect Freq - %s band', bandlist(bidx).name ), ...
    sprintf( 'combo-intersect-%s', bandlist(bidx).label ) );

end


disp(datetime);
disp('== Finished plotting.');




%
%
% Helper functions.


% Contour plot of confusion matrix stats vs magnitude and frequency threshold.

function wlHelper_plotContours( cfg, ...
  tpcounts, fpcounts, magrange, freqrange, ...
  tuples_mag, tuples_freq, figtitle, filelabel )


  % Matrix indices are (Y,X). Y = mag thresh, X = freq thresh.
  [ xgrid, ygrid ] = meshgrid( freqrange, magrange );


  % FIXME - Kludge levels.

%  countlevels = [ 0.3 0.5 0.7 1.0 1.5 2 3 5 7 10 15 20 30 50 70 100 ...
  countlevels = [ 1 2 3 5 7 10 15 20 30 50 70 100 ...
    150 200 300 500 700 1000 1500 2000 3000 5000 7000 10000 ...
    15000 20000 30000 50000 70000 100000 ];


  % True positive count.

  figure(cfg.fig);
  clf('reset');
```

```
contour(xgrid, ygrid, tpcounts, countlevels, 'ShowText', 'On');

xlabel('Frequency Threshold (dB)');
ylabel('Magnitude Threshold (dB)');

title(sprintf('%s - TP count', figtitle));
saveas( cfg.fig, sprintf('%s/contour-%s-tp.png', cfg.outdir, filelabel) );


% False positive count.

figure(cfg.fig);
clf('reset');

contour(xgrid, ygrid, fpcounts, countlevels, 'ShowText', 'On');

xlabel('Frequency Threshold (dB)');
ylabel('Magnitude Threshold (dB)');

title(sprintf('%s - FP count', figtitle));
saveas( cfg.fig, sprintf('%s/contour-%s-fp.png', cfg.outdir, filelabel) );


% False positive rates with thresholds annotated.

figure(cfg.fig);
clf('reset');

hold on;

fprates = fpcounts ./ (fpcounts + tpcounts);

contour(xgrid, ygrid, fprates, 'ShowText', 'On');

scatter(tuples_mag, tuples_freq, [], [ 0.9 0.2 0.3 ]);

hold off;

xlabel('Frequency Threshold (dB)');
ylabel('Magnitude Threshold (dB)');

title(sprintf('%s - FP rate', figtitle));
saveas( cfg.fig, ...
  sprintf('%s/contour-%s-fprate.png', cfg.outdir, filelabel) );


%
% Done.
```

```
end



% Segmentation by combined magnitude and frequency.
% "data" is the data trace to examine.
% "samprate" is the number of samples per second in the signal data.
% "bpfband" [min max] is the frequency band of interest. Edges may fade.
% "segconfig" is the segmentation algorithm configuration structure.
%   this contains the following fields:
%   "config_mag" is a segmentation configuration structure for
%     "wlProc_getEvSegmentsUsingMagDual()".
%   "config_freq" is a segmentation configuration structure for
%     "wlProc_getEvSegmentsUsingFreqDual()".
%   "thresh_mag" is an overriding value that replaces config_mag.dbpeak.
%   "thresh_freq" is an overriding value that replaces config_freq.dbpeak.
%   "operation" is "union" or "intersect", specifying how combined detection
%     is to occur.
%
% "events" is an array of event record structures per EVENTFORMAT.txt.
%   Only the following fields are provided:
%   "sampstart":  Sample index in "data" corresponding to burst nominal start.
%   "duration":   Time between burst nominal start and burst nominal stop.
%
% "waves" is a structure containing waveforms derived from "data":
%   "bpfwave" is the band-pass-filtered waveform.
%   "bpfmag", "bpffreq", and "bpfphase" are the analytic magnitude,
%     frequency, and phase of the bpf waveform.
%   "noisywave" is the noisy version of the bpf waveform.
%   "noisymag", "noisyfreq", and "noisyphase" are the analytic magnitude,
%     frequency, and phase of the noisy waveform.
%   "magpowerfast" is the rapidly-changing instantaneous power.
%   "magpowerslow" is the slowly-changing instantaneous power.
%   "fvarfast" is the rapidly-changing variance of the instantaneous frequency.
%   "fvarslow" is the slowly-changing variance of the instantaneous frequency.

function [ events, waves ] = wlHelper_getEvSegmentsUsingMagFreq( ...
  data, samprate, bpfband, segconfig )


  % FIXME - We don't have helper functions that return the detection vectors
  % directly, but we can compute them from the diagnostic waves, and then
  % run event detection manually.


  %
  % Do segmentation separately for magnitude and frequency.

  fnom = sqrt(bpfband(1) * bpfband(2));
```

```
[ events_mag, waves_mag ] = wlProc_getEvSegmentsUsingMagDual( ...
  data, samprate, bpfband, ...
  segconfig.config_mag.qglitch / fnom, ...
  segconfig.config_mag.qdrop / fnom, ...
  segconfig.config_mag.qlong / fnom, ...
  segconfig.thresh_mag, ...
  segconfig.config_mag.dbend );

[ events_freq, waves_freq ] = wlProc_getEvSegmentsUsingFreqDual( ...
  data, samprate, bpfband, ...
  segconfig.config_freq.noiseband, ...
  segconfig.config_freq.noisesnr, ...
  segconfig.config_freq.qglitch / fnom, ...
  segconfig.config_freq.qdrop / fnom, ...
  segconfig.config_freq.qlong / fnom, ...
  segconfig.config_freq.qshort / fnom, ...
  segconfig.thresh_freq, ...
  segconfig.config_freq.dbend );

% Merge wave structures.

waves = waves_freq;
waves.magpowerfast = waves_mag.magpowerfast;
waves.magpowerslow = waves_mag.magpowerslow;


%
% Calculate detection vectors.

magfactor = 10^(segconfig.thresh_mag / 10);
magend = 10^(segconfig.config_mag.dbend / 10);

% Looking for magnitude power greater than average.
detectmagpeak = waves.magpowerfast > (waves.magpowerslow * magfactor);
detectmagend = waves.magpowerfast > (waves.magpowerslow * magend);

% De-glitch the peak vector but not the end vector.
% Project the peak vector on to the end vector instead.
detectmagpeak = wlProc_calcDeGlitchedVector( detectmagpeak, ...
  round(samprate * segconfig.config_mag.qglitch / fnom), ...
  round(samprate * segconfig.config_mag.qdrop / fnom) );
detectmagend = detectmagend | detectmagpeak;


freqfactor = 10^(segconfig.thresh_freq / 10);
freqend = 10^(segconfig.config_freq.dbend / 10);

% Looking for frequency variance less than average.
detectfreqpeak = waves.fvarslow > (waves.fvarfast * freqfactor);
detectfreqend = waves.fvarslow > (waves.fvarfast * freqend);
```

```
% De-glitch the peak vector but not the end vector.
% Project the peak vector on to the end vector instead.
detectfreqpeak = wlProc_calcDeGlitchedVector( detectfreqpeak, ...
  round(samprate * segconfig.config_freq.qglitch / fnom), ...
  round(samprate * segconfig.config_freq.qdrop / fnom) );
detectfreqend = detectfreqend | detectfreqpeak;


% Merge the two detection vectors.

if strcmp('union', segconfig.operation)

  detectpeak = detectmagpeak | detectfreqpeak;
  detectend = detectmagend | detectfreqend;

else
  if ~strcmp('intersect', segconfig.operation)
    disp(sprintf( ...
      '### Unrecognized operation "%s"; falling back on "intersect".', ...
      operation ));
  end

  detectpeak = detectmagpeak & detectfreqpeak;
  detectend = detectmagend & detectfreqend;

end


%
% Get event segments from these detection vectors.

% Look for low-to-high transitions followed by high-to-low transitions in
% the endpoint detection vector.

evcount = 0;
foundlohi = false;

for sidx = 2:length(detectend)

  if detectend(sidx) && ~detectend(sidx-1)

    % This is the start of a potential event.
    foundlohi = true;
    lastlohi = sidx;  % Index of the first "true" element.

  elseif foundlohi && detectend(sidx-1) && ~detectend(sidx)

    % This is the end of a potential event (just past the end).
    % See if it has above-peak elements.
```

```
      peaksubset = detectpeak(lastlohi:sidx-1);
      if 0 < sum(peaksubset)

        % This is a real event.
        thisevent = struct( 'sampstart', lastlohi, ...
          'duration', (sidx - lastlohi) / samprate, ...
          'samprate', samprate );

        evcount = evcount + 1;
        events(evcount) = thisevent;
      end

    end

  end

  % Make sure we have a return value.

  if 1 > evcount
    events = [];
  end


  %
  % Done.

end



% Parameter extraction by incremental fitting.
%
% FIXME - Documentation goes here.

function [ newevents, auxdata ] = wlHelper_getEvParamsIncremental( ...
  oldevents, waves, samprate, paramconfig )

  % FIXME - NYI. Kludging with manual grid search.

  newevents = wlProc_getEvParamsUsingHilbert( waves.bpfwave, samprate, ...
    waves.bpfmag, waves.bpffreq, waves.bpfphase, ...
    oldevents, 7 );

  auxdata = struct();

end


%
```

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

E.2 do_ft_custom_detect.m

```
% Field Trip-related test scripts - Synthetic FT data - "Custom" - Detection.

%
%
% Includes.

% Assume that the parent called _init and _config.



%
%
% Event detection.


disp('== Detecting events using custom function.')
disp(sprintf( '-- (%dx%d patches)', ...
  length(threshsweep_xy), length(threshsweep_xy) ));
disp(datetime);


wavecomparefuncbpf = @(thisev, thiswave) ...
  wlProc_calcWaveErrorRelative( thiswave.bpfwave, thisev.sampstart, ...
    thisev.wave, thisev.s1 );
errpassfunc = @(thisev) (0.7 >= thisev.auxdata.errbpf);

clear detectft_union;
clear detectft_intersect;

patchtotal = length(threshsweep_xy) * length(threshsweep_xy);
patchcount = 0;

for thidxmag = 1:length(threshsweep_xy)
  for thidxfreq = 1:length(threshsweep_xy)

    patchcount = patchcount + 1;
    if tattlepatch
      disp(sprintf( '-- Checking XY sample %d of %d...', ...
        patchcount, patchtotal ));
    end


    for bidx = 1:length(bandlist)
```

```
      % FIXME - This might underrun "segdetendthresh" if ranges are strange.
      % Detection will still happen, but threshold is effectively clamped.

      threshmag = magthreshdb(bidx) + threshsweep_xy(thidxmag);
      threshfreq = freqthreshdb(bidx) + threshsweep_xy(thidxfreq);

      bandoverrides(bidx) = struct ( ...
        'seg', ...
          struct( 'thresh_mag', threshmag, 'thresh_freq', threshfreq ), ...
        'param', struct() );

      % FIXME - We can't override DC LPF in the nested configuration.
      % Well, we can, but we have to copy the entire daughter structure.

    end


    detect_temp = wlFT_doFindEventsInTrials_MT(traceft, bandlist, ...
      segconfig_union, paramconfig_increment, ...
      bandoverrides, tattledetect );
    detect_temp = wlFT_pruneEventsByTime(detect_temp, padtime, padtime);
    detect_temp = ...
      wlFT_calcEventErrors(detect_temp, wavecomparefuncbpf, 'errbpf');
    detect_temp = wlAux_pruneMatrix(detect_temp, errpassfunc);

    detectft_union{thidxmag, thidxfreq} = detect_temp;

    detect_temp = wlFT_doFindEventsInTrials_MT(traceft, bandlist, ...
      segconfig_intersect, paramconfig_increment, ...
      bandoverrides, tattledetect );
    detect_temp = wlFT_pruneEventsByTime(detect_temp, padtime, padtime);
    detect_temp = ...
      wlFT_calcEventErrors(detect_temp, wavecomparefuncbpf, 'errbpf');
    detect_temp = wlAux_pruneMatrix(detect_temp, errpassfunc);

    detectft_intersect{thidxmag, thidxfreq} = detect_temp;

  end
end


disp(datetime);
disp('== Finished detecting events.');



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