Offline Analysis Tools

Working with TDT data in Matlab

Updated 2024-03-13

Tucker-Davis Technologies
11930 Research Circle
Alachua, FL 32615 USA
Phone: +1.386.462.9622
Fax: +1.386.462.5365

Notices

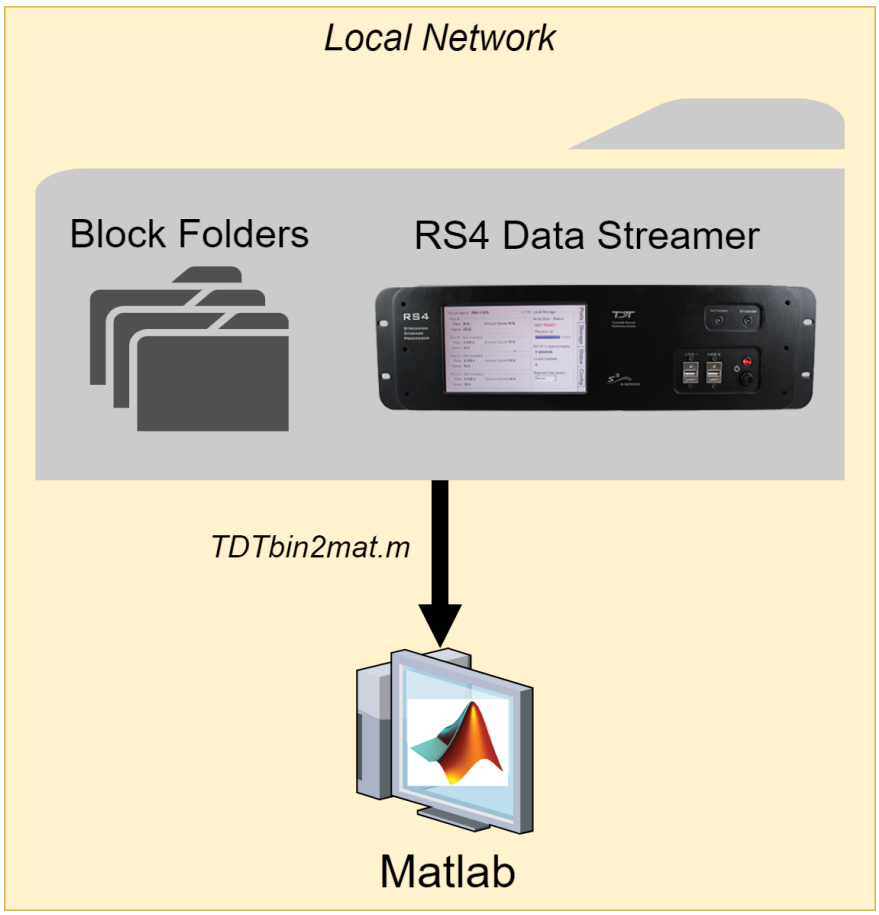
The information contained in this document is provided "as is," and is subject to being changed, without notice. TDT shall not be liable for errors or damages in connection with the furnishing, use, or performance of this document or of any information contained herein.

The latest versions of TDT documents are always online at https://www.tdt.com/docs/

Table of Contents

Overview of MATLAB Offline Analysis Tools

Read block files directly from disk or SEV files directly from RS4.



See TDT Data Storage for a description of the folder structure.

TDTbin2mat

`TDTbin2mat` is an all-in-one function for reading TDT data into MATLAB. It needs only one input: the block path.

```
data = TDTbin2mat('C:\TDT\Synapse\Tanks\Exp1-160921-120606\Sub1-1');
```

`TDTbin2mat` will return a structure containing all recorded data from that block, organized by type. See TDT Data Types for a description of the data types.

`TDTbin2mat` uses parameter value combinations to refine the imported data. To extract specific event types only, use the `'TYPE'` option. For example, to import epocs and snippets only, use this:

```
data = TDTbin2mat('C:\TDT\TDTExampleData\Algernon-180308-130351', ...
    'TYPE', {'epocs', 'snips'})
```

Use the `'STORE'` option to extract a particular data store by name, in this example a streaming event called `'Wav1'`. Combine this with the `'CHANNEL'` option to extract a single channel, or list of channels, in this case channel 2:

```
data = TDTbin2mat('C:\TDT\TDTExampleData\Algernon-180308-130351', ...
    'STORE', 'Wav1', 'CHANNEL', 2);
```

You can also filter by time, if you are only interested in portions of the recording, or if the entire recording won't fit into available memory (RAM) at one time. Use the `'T1'` and `'T2'` options to specify the start and stop time, in seconds, to retrieve from the block. This example reads only from time T1=10s to time T2=20s of the block into MATLAB:

```
data = TDTbin2mat('C:\TDT\TDTExampleData\Algernon-180308-130351', ...
    'T1', 10, 'T2', 20);
```

`TDTbin2mat` offers many more useful options that are described in its help documentation.

```
>> help TDTbin2mat
```

TDTfilter

`TDTfilter` applies advanced epoc filtering to extracted data. For example, if you only want to look at data around a certain epoc event, you will use `TDTfilter` to do this. See the Raster/PSTH example for a complete demonstration.

To only look at data around the epoc event timestamps, use a `'TIME'` filter. In this example, data from 20 ms before the *Levl* epoc to 50ms after the onset is retained.

```
data = TDTfilter(data, 'Levl', 'TIME', [-0.02, 0.07]);
```

To only look at data when an epoc was a certain value, use a `'VALUES'` filter. In this example, only data when the *Freq* epoc was equal to 9000 or 10000 is retained.

```
data = TDTfilter(data, 'Freq', 'VALUES', [9000, 10000]);
```

If you want to look for a particular behavioral response that occurs sometime during the allowed time range, use the `'MODIFIERS'` filter. In this example, only data when the *Freq* epoc was 10000 **AND** the *Resp* epoc had a value of 1 sometime during the *Freq* epoc is retained.

```
data = TDTfilter(data, 'Freq', 'VALUES', [10000]);
data = TDTfilter(data, 'Resp', 'MODIFIERS', [1]);
```

As you can see, for complex filtering the output from one call to `TDTfilter` can become the input to the next call to `TDTfilter`. If your data sets are large, or if you are iterating through many combinations of epoc variables, it is preferred to extract only the epocs and do all of the epoc filtering first to find the valid time ranges that match the filter, and then use this as the `'RANGES'` input to `TDTbin2mat` to extract all events (including snips, streams) on only those valid time ranges.

```
% read just the epoc events
data = TDTbin2mat(block_path, 'TYPE', {'epocs'});

% use the epocs to find time ranges we want
data = TDTfilter(data, 'Freq', 'VALUES', [9000, 10000]);
data = TDTfilter(data, 'Levl', 'VALUES', [70, 80, 90]);
data = TDTfilter(data, 'Resp', 'MODIFIERS', [1]);

% read just the value time ranges from the whole data set
data = TDTbin2mat(block_path, 'RANGES', data.time_ranges);
```

SEV2mat

`SEV2mat` reads SEV files into a MATLAB structure. SEV files are created by the RS4 Data Streamer or by enabling the Discrete Files option when streaming continuous signals in Synapse. SEV files consist of a single channel of data per file, with a short header, so it is very fast to read them. `TDTbin2mat` will automatically call `SEV2mat` if it finds SEV files in the block directory. Like `TDTbin2mat`, `SEV2mat` needs only one input: the block path.

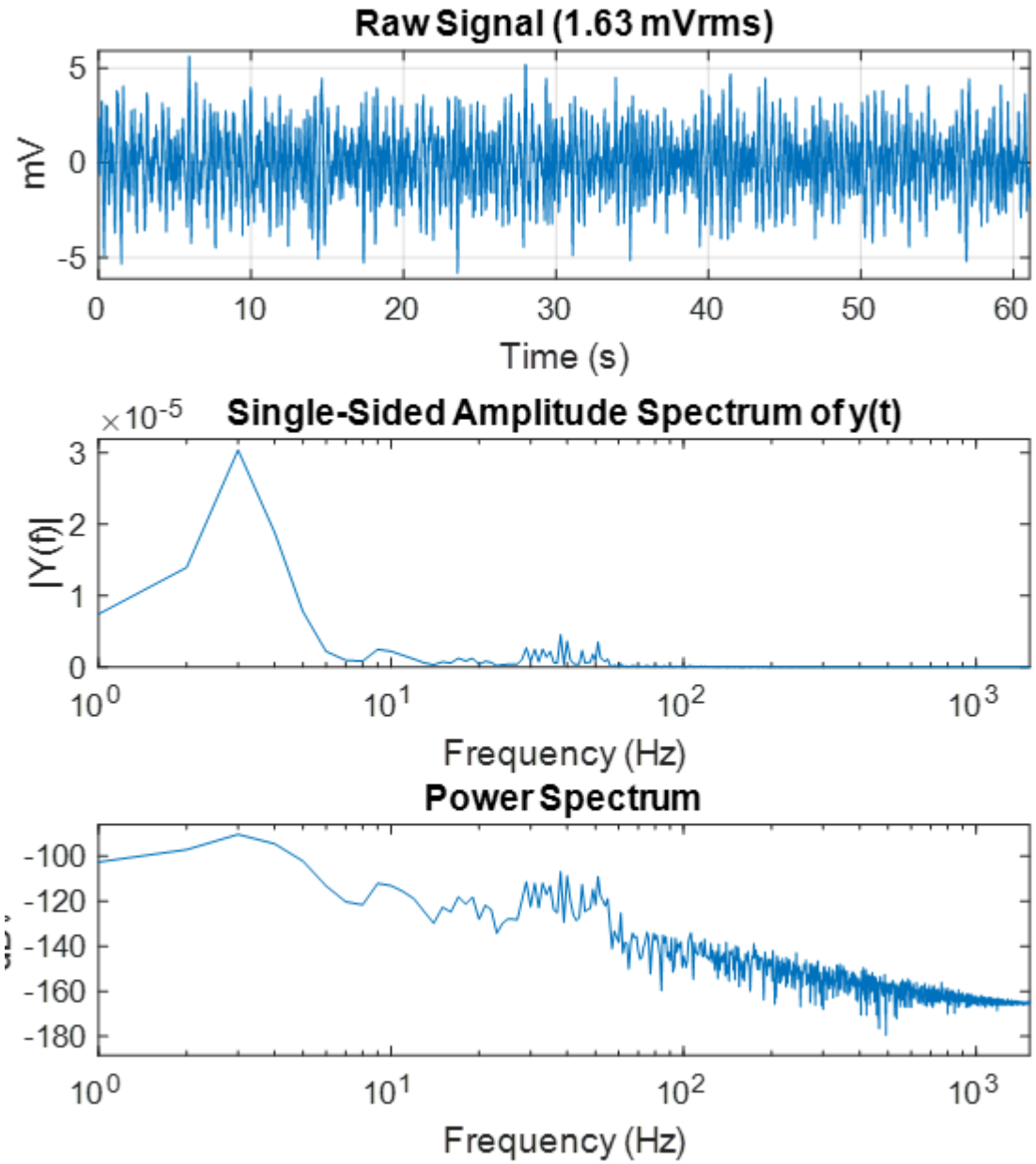
```
data = SEV2mat('C:\TDT\Synapse\Tanks\Exp1-160921-120606\Sub1-1');
```

`SEV2mat` will return a structure containing the streams that it found. Each stream field includes the data array and sampling rate.

TDTfft

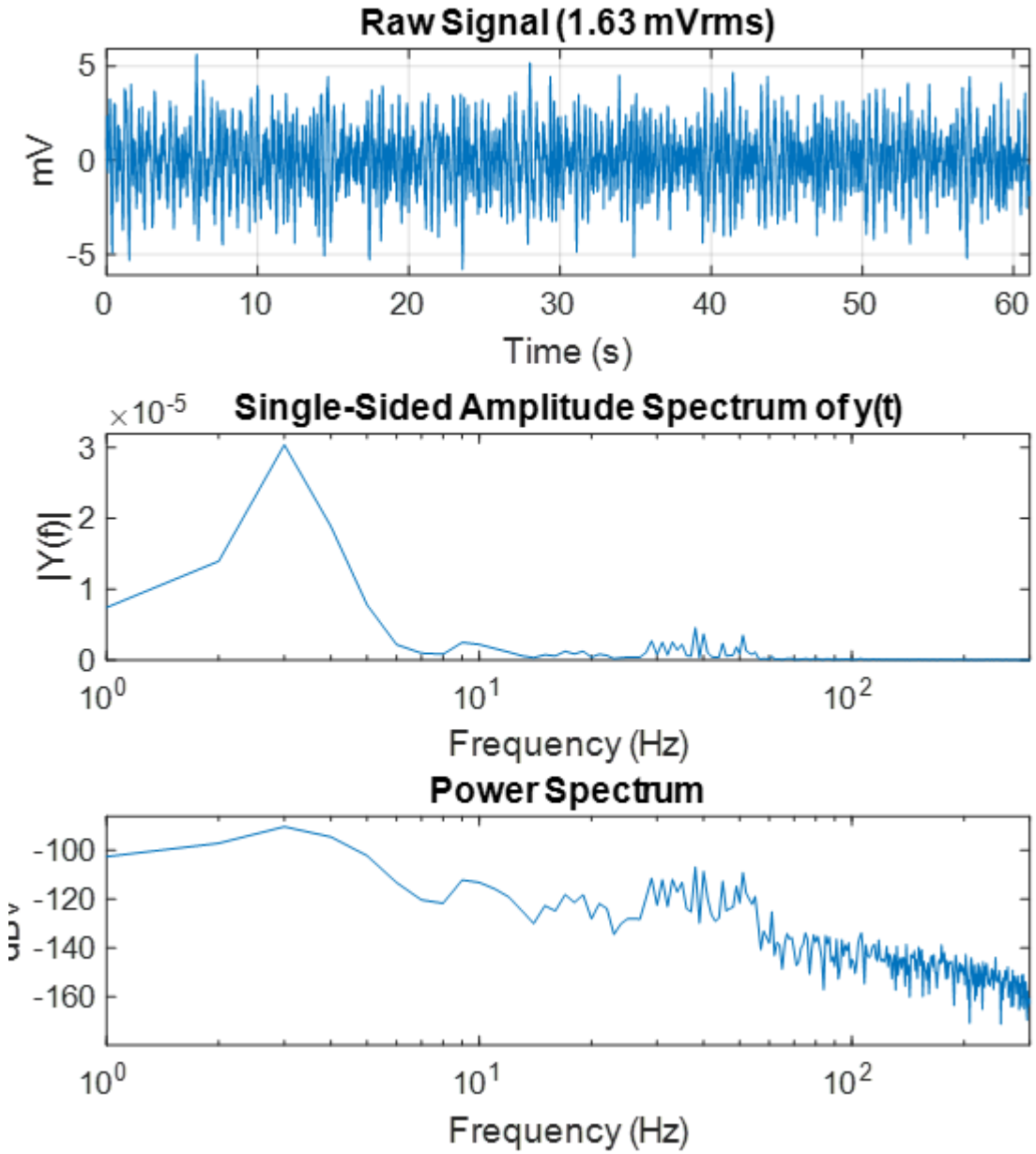
`TDTfft` performs frequency analysis on the data stream. It takes the stream data output of `TDTbin2mat` and a channel number as inputs. In this example, it is plotting channel 1 of the `'LFP1'` store from an example block.

```
data = TDTbin2mat('C:\TDT\TDTExampleData\Algernon-180308-130351');
TDTfft(data.streams.LFP1, 1);
```



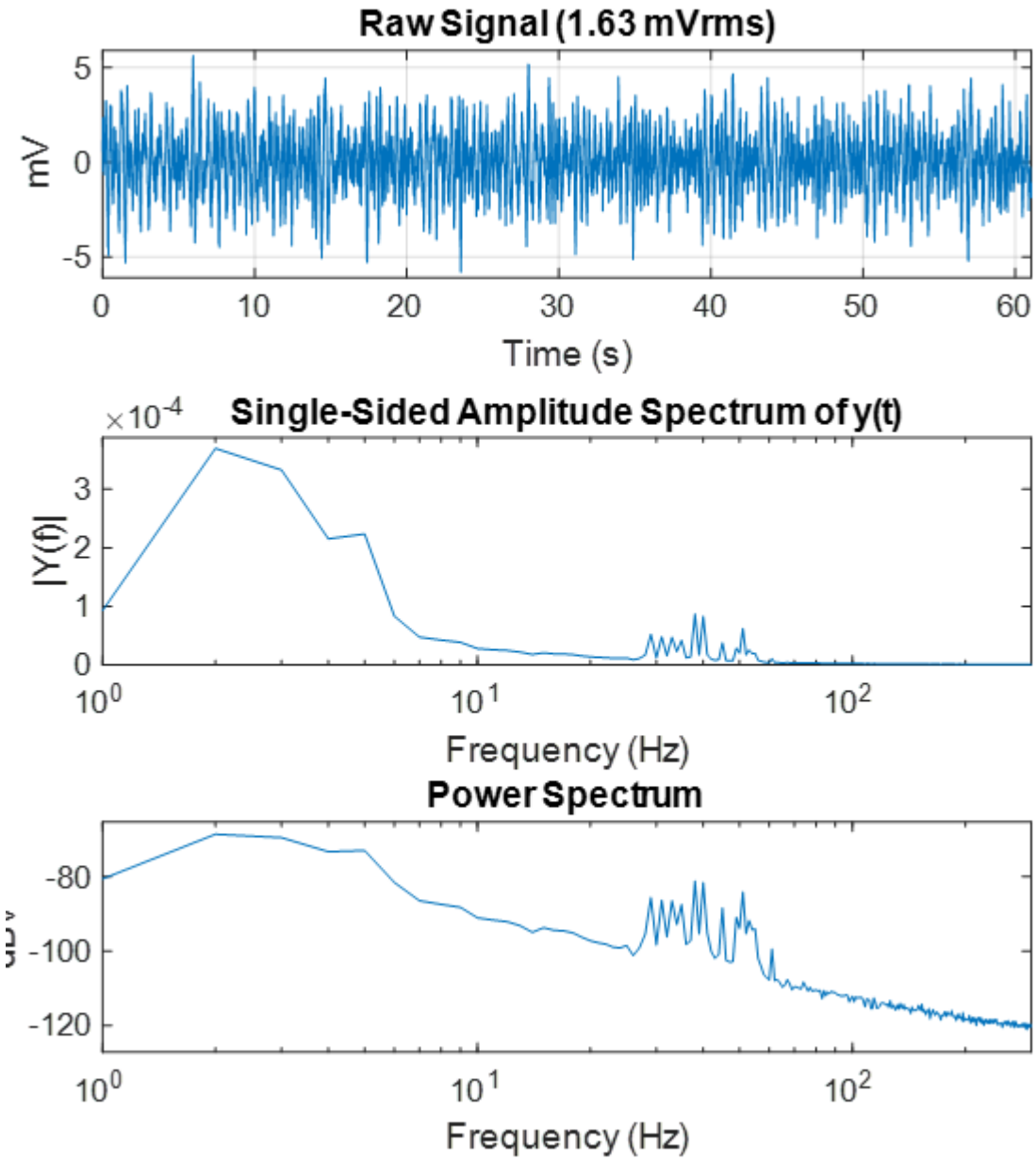
Use the `'FREQ'` input to specify the frequency range that you're interested in.

```
% look at frequencies 0 Hz to 300 Hz only
TDTfft(data.streams.LFP1, 1, 'FREQ', [0, 300]);
```



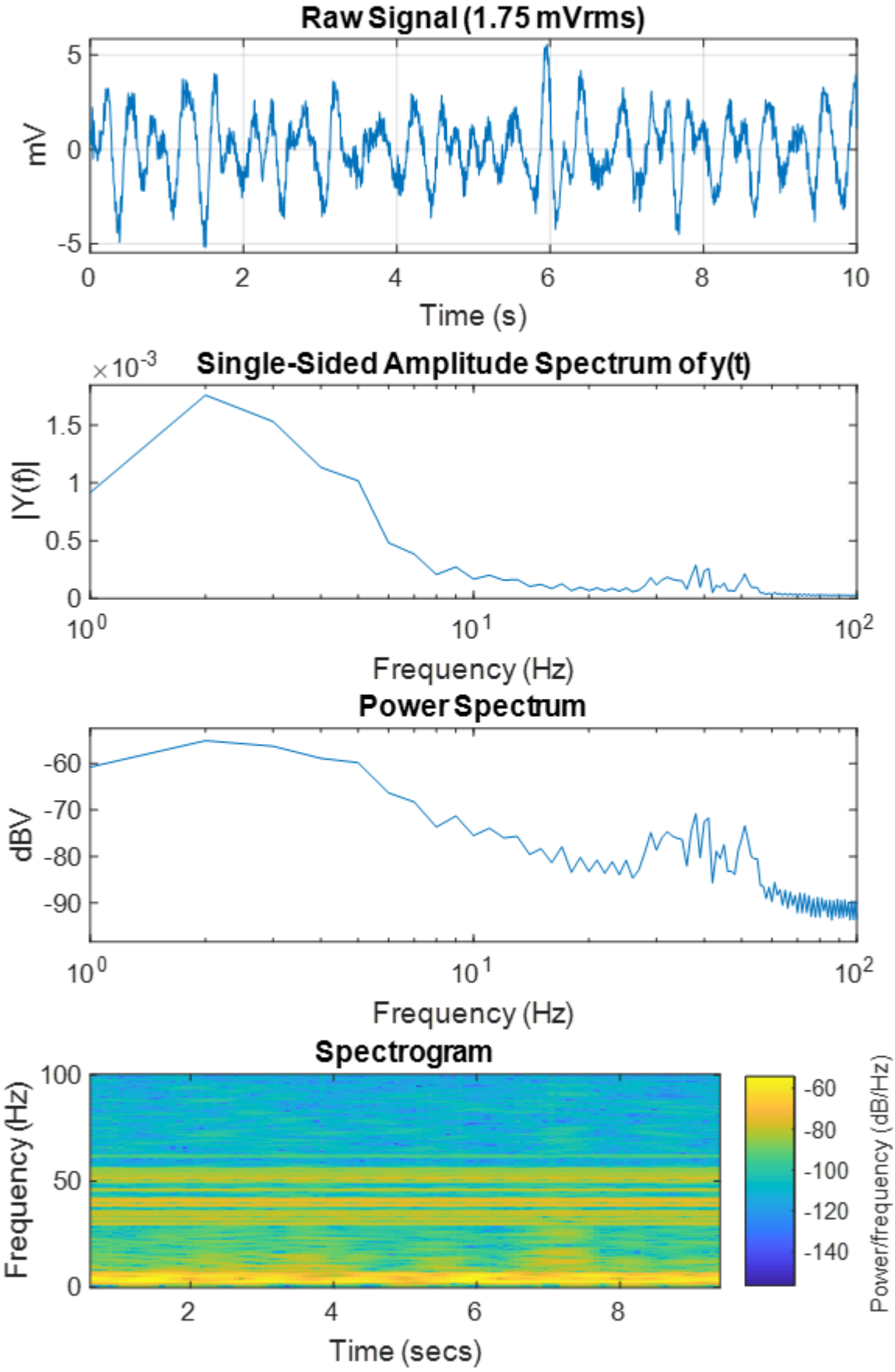
Often the frequency plot is noisy. Use the `'NUMAVG'` input to smooth out the frequency plot and better isolate the components with the most power through the entire recording. This breaks the data into chunks, performs the FFT on those chunks, and then plots the average.

```
% split data up into 20 chunks and plot average FFT
TDTfft(data.streams.LFP1, 1, 'FREQ', [0, 300], 'NUMAVG', 20);
```



Set the `'SPECPLOT'` input to 1 to also display a spectrogram of the entire data stream. If the data set is too long then MATLAB could run out of memory to do this, and you would need to use the `T1` and `T2` parameters in `TDTbin2mat` to look at smaller chunks of data.

```
data = TDTbin2mat('C:\TDT\TDTExampleData\Algernon-180308-130351', 'T2', 10);
TDTfft(data.streams.LFP1, 1, 'FREQ', [0, 100], 'NUMAVG', 20,'SPECPLOT', 1);
```

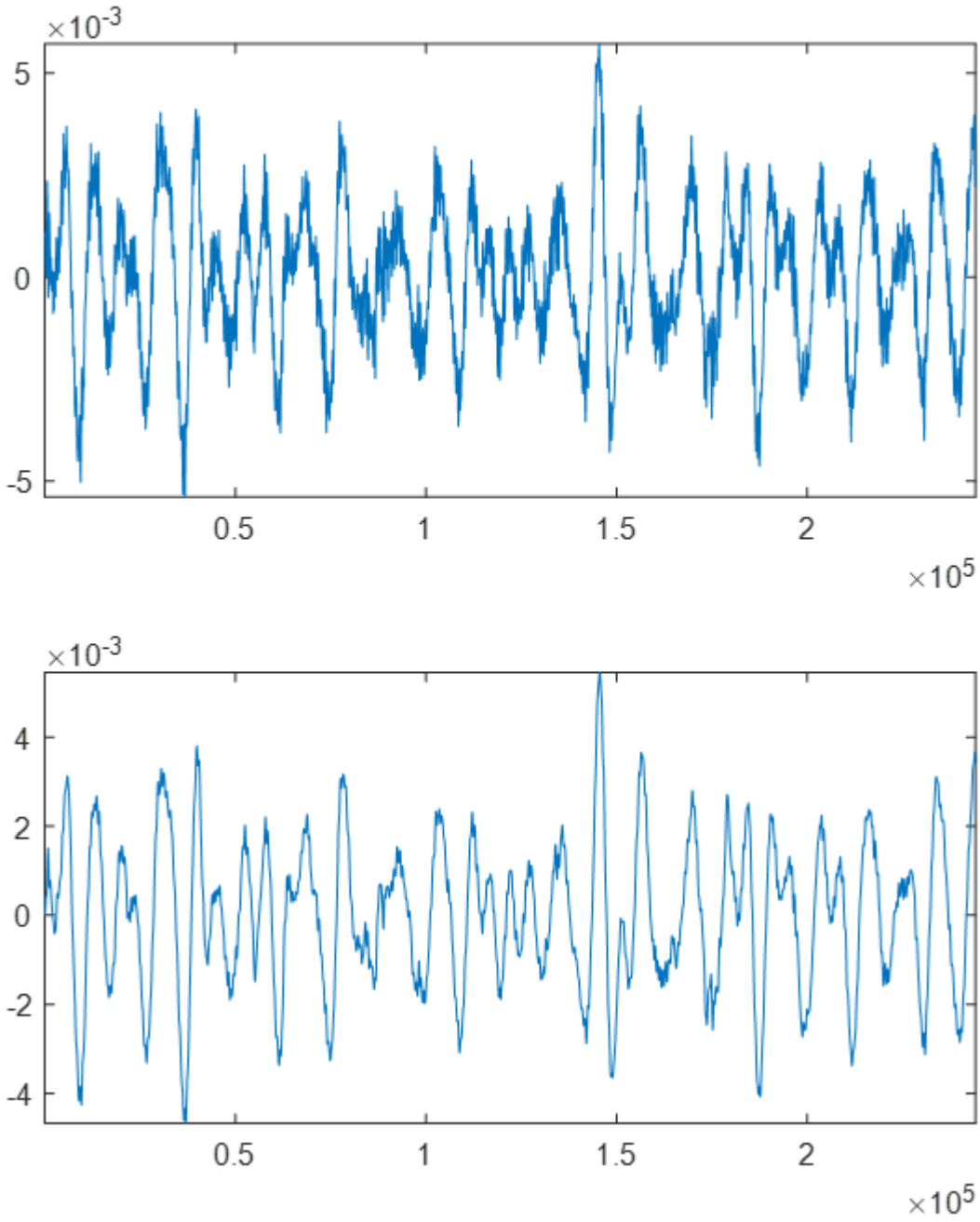


See the LFP plot example for a full demonstration.

TDTdigitalfilter

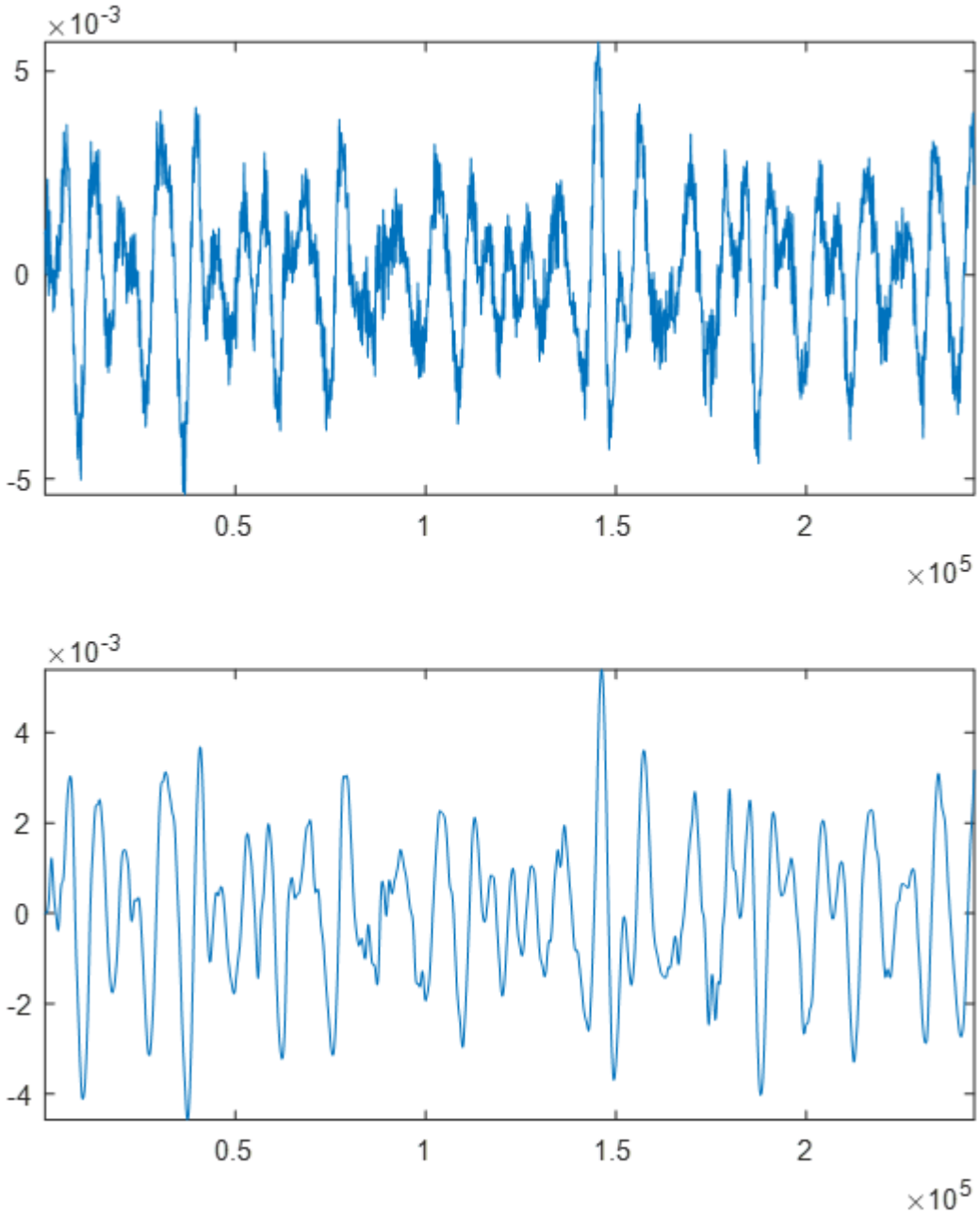
`TDTdigitalfilter` mimics the hardware digital filters in the Neural Stream Processor gizmo. It takes the output of `TDTbin2mat` and applies the specified digital filter to the specified streaming data store. Here is a simple DC-20 Hz bandpass filter example:

```
data = TDTbin2mat('C:\TDT\TDTExampleData\Algernon-180308-130351', 'T2', 10);
subplot(2,1,1);
plot(data.streams.Wav1.data(1,:));
data = TDTdigitalfilter(data, 'Wav1', [0, 20]); % look at 0-20 Hz only
subplot(2,1,2);
plot(data.streams.Wav1.data(1,:));
```



Use the `'ORDER'` input to set the filter order. By default it is a 2 for a second order filter.

```
data = TDTdigitalfilter(data, 'Wav1', [0, 20], 'ORDER', 8);
```



You can also add a notch filter.

```
data = TDTdigitalfilter(data, 'Wav1', 'NOTCH', 60, 'ORDER', 4);
```

TDTthresh

`TDTthresh` takes continuous data and applies a thresholding algorithm to extract snippets. It has two modes, `'manual'` and `'auto'`. In `'manual'` mode, supply the absolute threshold input `'THRESH'` to extract snippets. Can be negative for negative-first spike detection.

In `'auto'` mode, a multiple of the sliding RMS window is used to calculate the instantaneous threshold. Set the `'TAU'` parameter to specify the time window, and set the `'STD'` parameter to set the scalar. So if TAU=5 and STD=6, then the threshold will track 6*RMS of the previous 5 seconds.

The new snippet store generated from `TDTthresh` is called `'Snip'`.

```
data = TDTbin2mat('C:\TDT\TDTExampleData\Algernon-180308-130351', 'T2', 10);
data = TDTdigitalfilter(data, 'Wav1', [300 5000]);
data = TDTthresh(data, 'Wav1', 'MODE', 'auto');
subplot(2,1,1);
plot(data.streams.Wav1.data(1,:));
subplot(2,1,2);
plot(data.snips.Snip.data');
```

You can set the `'TETRODE'` flag to extract snippets organized by groups of 4 channels. If the threshold crosses on one channel, all 4 channels will get a snippet.

`TDTthresh` has a few more customizations you can explore in its help documentation.

```
>> help TDTthresh
```

See the Rethreshold Example for a complete demonstration of `TDTdigitalfilter` and `TDTthresh`.

TDT Data Storage

Data collected or used by TDT software is stored in tanks - special directories on your hard drive. Each time you press 'Record' in Synapse, a new block is created within the tank. Within a block different stores can record different types of events at different rates. The blocks are special folders within the tank directories.

TDT Data Types

1. **epocs** are values stored with onset and offset timestamps that can be used to create time-based filters on your data. They can be created by the Epoch Data Storage gizmo, Logic gizmos, Stimulation gizmos, and many more.
    a. If Runtime Notes were enabled in Synapse, they will appear in `data.epocs.Note`. The notes themselves will be in `data.epocs.Note.notes`.

    See the Synapse Manual for more information on Runtime Notes.
2. **streams** are continuous single channel or multichannel recordings, like those stored by the Stream Data Storage gizmo, the Fiber Photometry gizmo, and many others. The structure includes the data array and sampling rate.
3. **snips** are short snippets of data collected on a trigger. For example, action potentials recorded around threshold crossings in the Spike Sorting gizmos, or fixed duration snippets recorded by the Strobe Store gizmo. This structure includes the waveforms, channel numbers, sort codes, trigger timestamps, and sampling rate.
4. **scalars** are similar to epocs but can be single or multi-channel values and only store an onset timestamp when triggered. These can be created by the Strobe Store gizmo.

The returned structure also contains an **info** field with block start/stop times, duration, and information about the Subject, User, and Experiment that it came from (if the block was created in Synapse).

Merging Blocks

There is a tool called TankManager that installs with Synapse that allows you to concatenate blocks directly on disk. This can create a 'super block' by combining the snippets / epocs events from multiple blocks into a single block before you export the data to another format or import to OpenSorter. It shifts the timestamps of the second block (it adds ~10 second gap in between the recordings so there is no overlap).

TankManager is a command line executable. The format for merging blocks is:

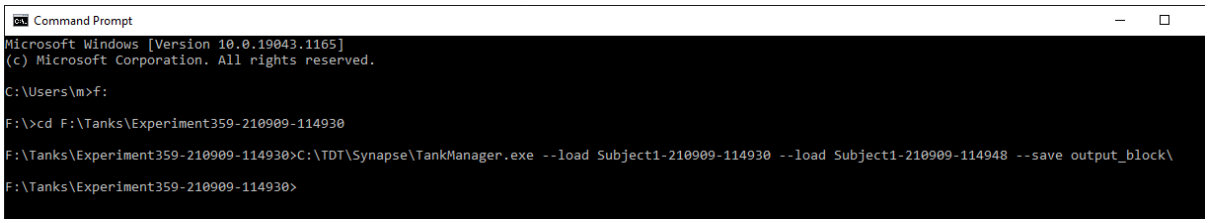
```
C:\TDT\Synapse\TankManager.exe --load block1 --load block2 --save outputblock
```

Replace the text with the two block names that you want to merge.

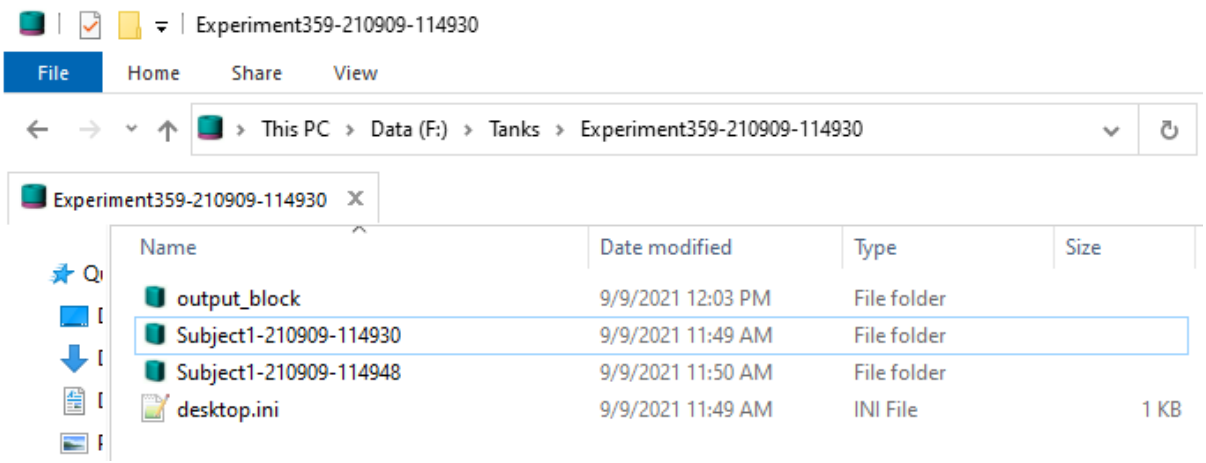
Here is an example that merges the two blocks called `Subject1-210909-114930` and `Subject1-210909-114948` in the tank `F:\Tanks\Experiment359-210909-114930` into a new block called `output_block` inside that same tank:

1. Press the Windows key (or Windows + R) and type `cmd` to enter the command line.
2. Change directory into your tank folder.
3. Run the TankManager command. In this example, it is

```
C:\TDT\Synapse\TankManager.exe --load "Subject1-210909-114930" --load
"Subject1-210909-114948" --save "output_block"
```



This makes a new block called `output_block` inside the same tank folder, which you can then use like any other block.



Splitting Blocks

You can extract time ranges of blocks and export them to smaller blocks using the same TankManager utility described above. This works best with snippet, epoc, and scalar data types.

Here's an example that will keep the data from time t=5s to t=10s from the block `C:\BLOCKPATH` and write it to a new block called `C:\BLOCKPATH_5_10`:

```
C:\TDT\Synapse\TankManager -l "C:\BLOCKPATH" -k "5-10" -s "C:\BLOCKPATH_5_10"
```

For example, you could use this to split up your blocks into hour long chunks like this:

```
C:\TDT\Synapse\TankManager -l "C:\BLOCKPATH" -k "0-3600" -s "C:\BLOCKPATH_0hr"
C:\TDT\Synapse\TankManager -l "C:\BLOCKPATH" -k "3600-7200" -s "C:
\BLOCKPATH_1hr"
```

And so on. The data in these new blocks won't start until the first timestamp, so for later blocks there will be a large gap in the beginning. If you have an issue reading it with OpenSorter or another of our data utilities you might try following the steps in Tech Note TN0909.

> **Note**
>
> You might get a message during the block creation that says `Failed to set the system attribute on the block`. It is okay to ignore this.

Importing TDT Data into MATLAB for Offline Analysis

TDT provides a set of cross-platform tools for reading TDT data files directly into MATLAB offline. It is compatible with MATLAB 2007b and greater.

Requirements

1. MATLAB 2007b and greater

Installation

Just follow these three steps to get started:

1. Download the MATLAB SDK (zip) and extract the files into a directory on your computer, like `C:\TDT\TDTMatlabSDK`.
2. Download Example Data. Extract the example data files into the 'Examples' folder in the SDK.
3. Add the path to the unzipped MATLAB tools directory.

```
SDKPATH = 'C:\TDT\TDTMatlabSDK'; % or whatever path you extracted the SDK
zip into
addpath(genpath(SDKPATH));
```

4. Importing data is easy

```
data = TDTbin2mat('path/to/block');
```

Example Workbooks

Averaging Example

Import stream and epoc data into MATLAB using TDTbin2mat
Plot the average waveform around the epoc event
Good for Evoked Potential detection

Download M File

Housekeeping

Clear workspace and close existing figures. Add SDK directories to MATLAB path.

```
close all; clear all; clc;
[MAINEXAMPLEPATH,name,ext] = fileparts(cd); % \TDTMatlabSDK\Examples
DATAPATH = fullfile(MAINEXAMPLEPATH, 'ExampleData'); %
\TDTMatlabSDK\Examples\ExampleData
[SDKPATH,name,ext] = fileparts(MAINEXAMPLEPATH); % \TDTMatlabSDK
addpath(genpath(SDKPATH));
```

Importing the Data

This example assumes you downloaded our example data sets and extracted it into the \TDTMatlabSDK\Examples directory. To import your own data, replace 'BLOCKPATH' with the path to your own data block.

In Synapse, you can find the block path in the database. Go to Menu → History. Find your block, then Right-Click → Copy path to clipboard.

```
BLOCKPATH = fullfile(DATAPATH,'Algernon-180308-130351');
```

Set up the variables for the data you want to extract. We will extract channel 3 from the LFP1 stream data store, created by the Neural Stream Processor gizmo, and use our PulseGen epoc event ('PC0/') as our stimulus onset.

```
REF_EPOC = 'PC0/';
STREAM_STORE = 'LFP1';
CHANNEL = 3;
ARTIFACT = Inf; % optionally set an artifact rejection level
TRANGE = [-0.3, 0.8]; % window size [start time relative to epoc onset,
window duration]
```

Now read the specified data from our block into a MATLAB structure.

```
data = TDTbin2mat(BLOCKPATH, 'TYPE', {'epocs', 'scalars', 'streams'},
'CHANNEL', CHANNEL);
```

```
read from t=0.00s to t=61.23s
```

Use TDTfilter to extract data around our epoc event

Using the 'TIME' parameter extracts data only from the time range around our epoc event. For stream events, the chunks of data are stored in cell arrays.

```
data = TDTfilter(data, REF_EPOC, 'TIME', TRANGE);
```

Optionally remove artifacts.

```
art1 = ~cellfun('isempty', cellfun(@(x) x(x>ARTIFACT), data.streams.
(STREAM_STORE).filtered, 'UniformOutput',false));
art2 = ~cellfun('isempty', cellfun(@(x) x(x<-ARTIFACT), data.streams.
(STREAM_STORE).filtered, 'UniformOutput',false));
good = ~art1 & ~art2;
data.streams.(STREAM_STORE).filtered = data.streams.
(STREAM_STORE).filtered(good);
numArtifacts = sum(~good);
```

Applying a time filter to a uniformly sampled signal means that the length of each segment could vary by one sample. Let's find the minimum length so we can trim the excess off before calculating the mean.

```
minLength = min(cellfun('prodofsize', data.streams.(STREAM_STORE).filtered));
data.streams.(STREAM_STORE).filtered = cellfun(@(x) x(1:minLength),
data.streams.(STREAM_STORE).filtered, 'UniformOutput',false);
```

Find the average signal.

```
allSignals = cell2mat(data.streams.(STREAM_STORE).filtered');
meanSignal = mean(allSignals);
stdSignal = std(double(allSignals));
```

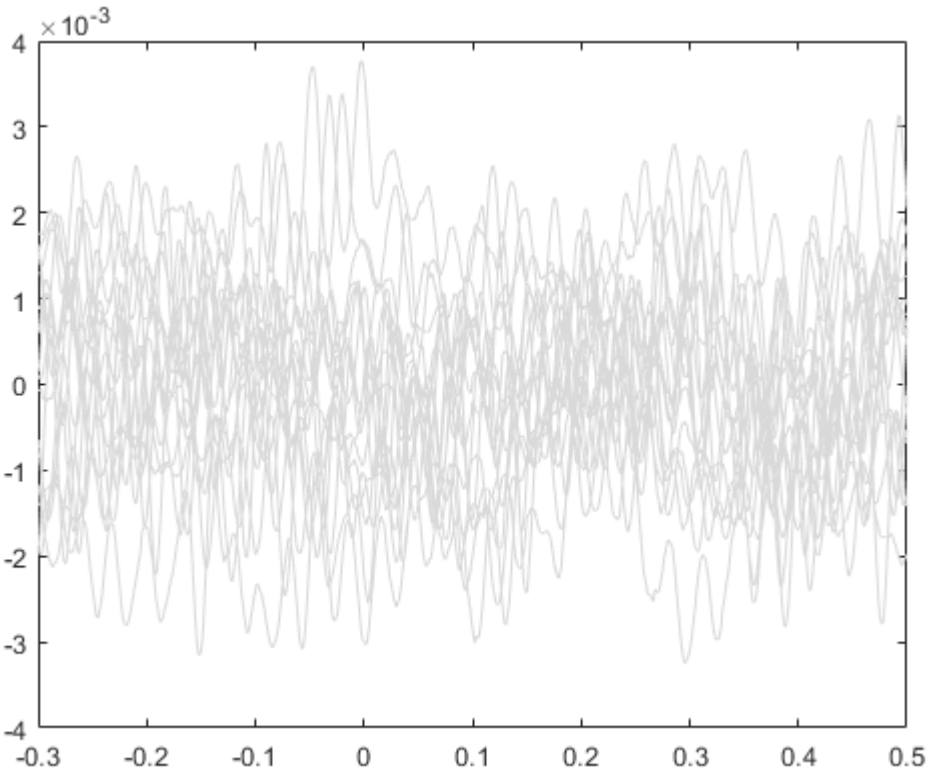
Ready to plot

Create the time vector.

```
ts = TRANGE(1) + (1:minLength) / data.streams.(STREAM_STORE).fs;
```

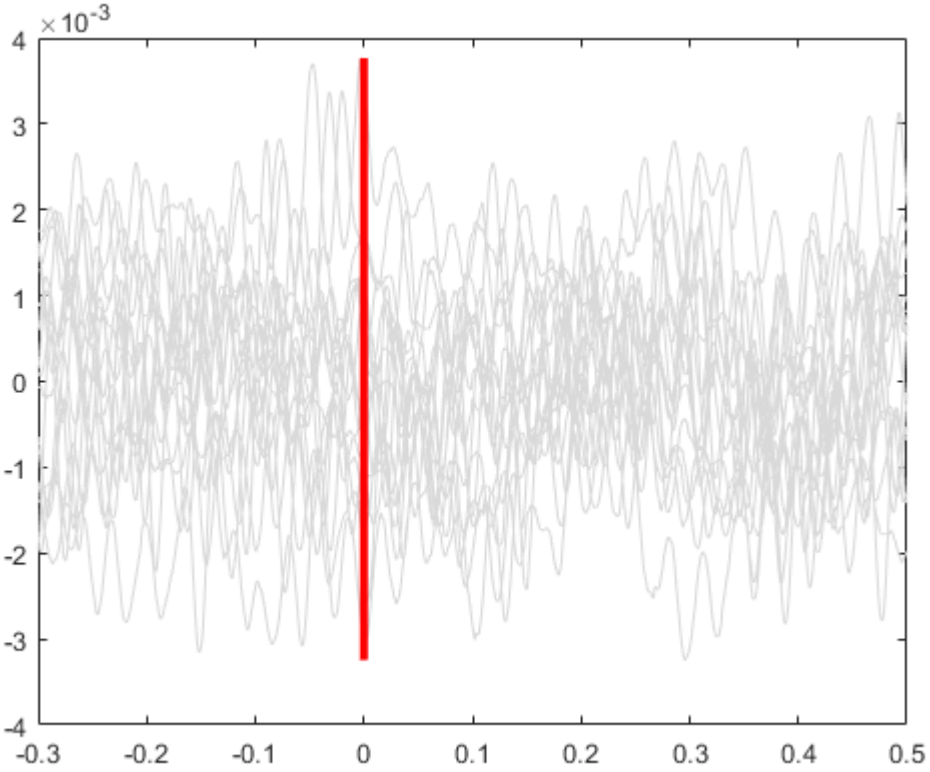
Plot all the signals as gray.

```
plot(ts, allSignals','Color', [.85 .85 .85]); hold on;
```



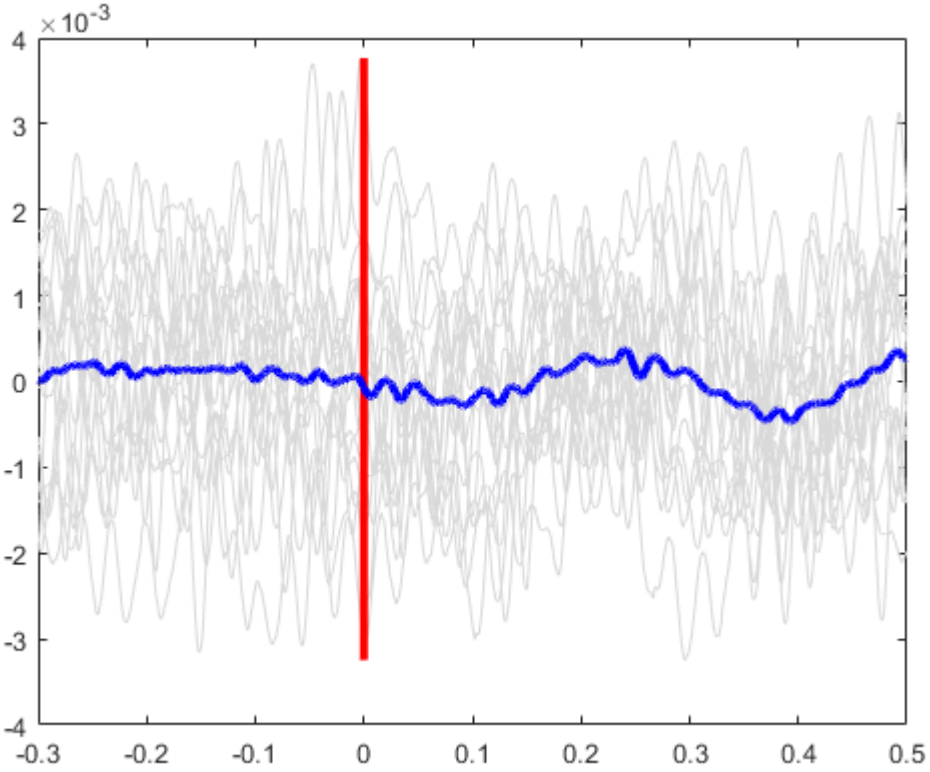
Plot vertical line at time=0.

```
line([0 0], [min(allSignals(:)), max(allSignals(:))], 'Color', 'r',
'LineStyle','-', 'LineWidth', 3)
```



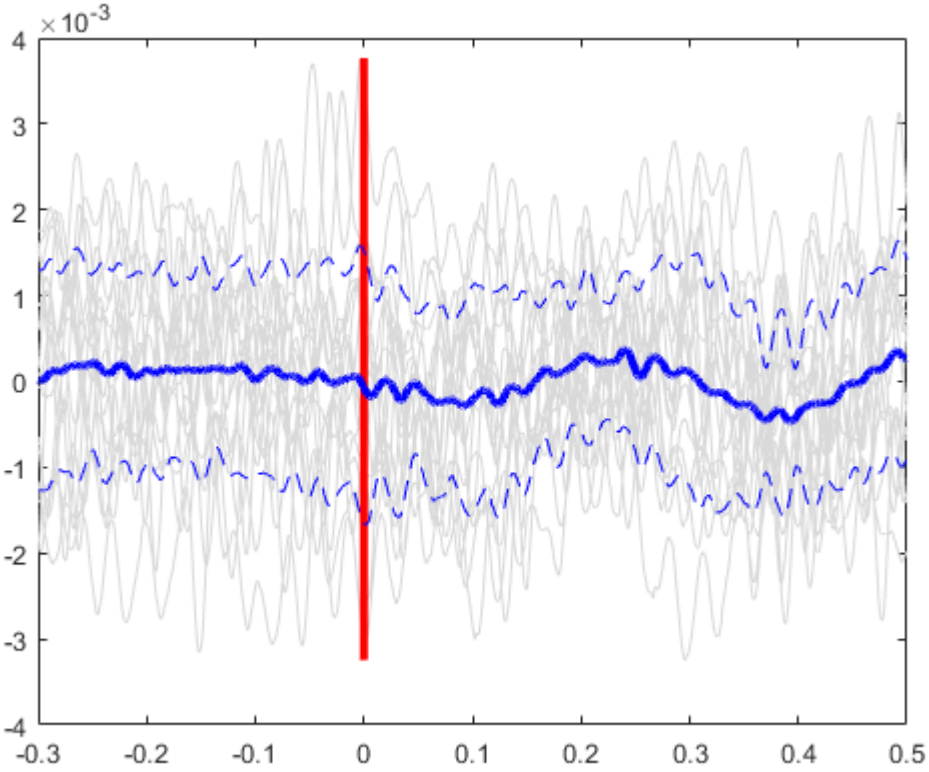
Plot the average signal.

```
plot(ts, meanSignal, 'b', 'LineWidth', 3)
```



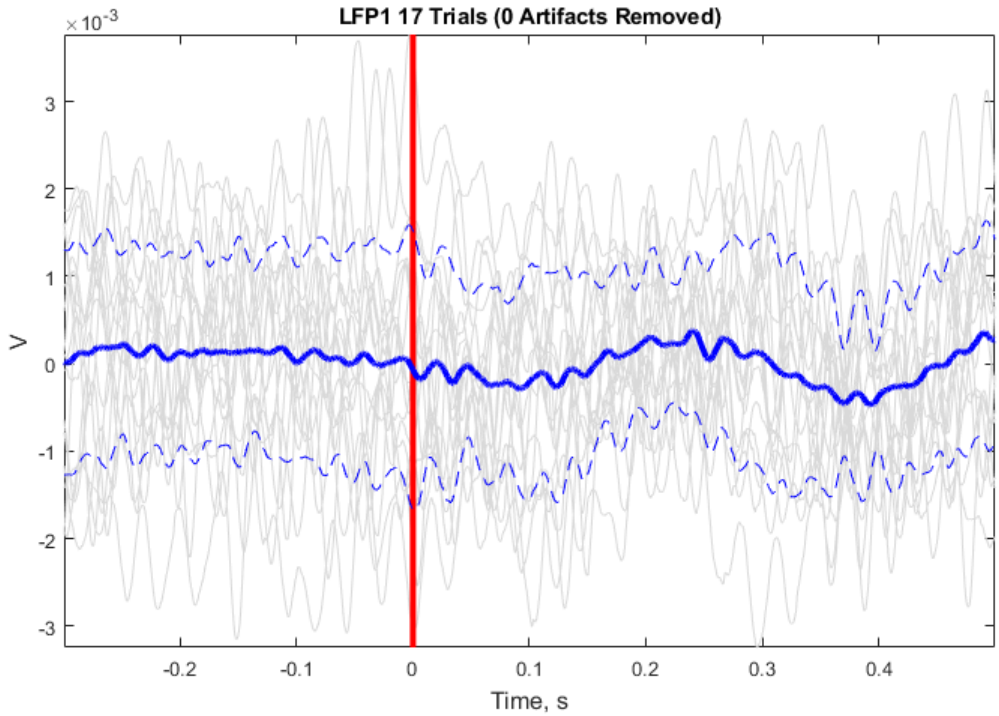
Plot the standard deviation bands.

```
plot(ts, meanSignal+stdSignal, 'b--', ts, meanSignal-stdSignal, 'b--');
```



Finish up the plot

```
axis tight
xlabel('Time, s','FontSize',12)
ylabel('V', 'FontSize', 12)
title(sprintf('%s %d Trials (%d Artifacts Removed)', STREAM_STORE,
numel(data.streams.(STREAM_STORE).filtered), numArtifacts))
set(gcf, 'Position',[100, 100, 800, 500])
```


LFP1 17 Trials (0 Artifacts Removed)
×10^-3
3
2
1
0
-1
-2
-3
V
-0.2
-0.1
0
0.1
0.2
0.3
0.4
Time, s

LFP Plot Example

Import streaming LFP data into MATLAB using TDTbin2mat
Plot the power spectrum and RMS of the waveform
Good for sleep scoring and behavioral discrimination

Download M File

Housekeeping

Clear workspace and close existing figures. Add SDK directories to MATLAB path.

```
close all; clear all; clc;
[MAINEXAMPLEPATH,name,ext] = fileparts(cd); % \TDTMatlabSDK\Examples
DATAPATH = fullfile(MAINEXAMPLEPATH, 'ExampleData'); %
\TDTMatlabSDK\Examples\ExampleData
[SDKPATH,name,ext] = fileparts(MAINEXAMPLEPATH); % \TDTMatlabSDK
addpath(genpath(SDKPATH));
```

Importing the Data

This example assumes you downloaded our example data sets and extracted it into the \TDTMatlabSDK\Examples directory. To import your own data, replace 'BLOCKPATH' with the path to your own data block.

In Synapse, you can find the block path in the database. Go to Menu → History. Find your block, then Right-Click → Copy path to clipboard.

```
BLOCKPATH = fullfile(DATAPATH,'Algernon-180308-130351');
```

Let's pick the store name, channel, and how much data we want to extract, in seconds.

```
STORE = 'LFP1';
CHANNEL = 1;
T2 = 10;
```

Now read channel 1 from all stream data into a MATLAB structure called 'data'.

```
data = TDTbin2mat(BLOCKPATH, 'TYPE', {'streams'}, 'STORE', STORE, 'CHANNEL',
CHANNEL, 'T2', T2);
```

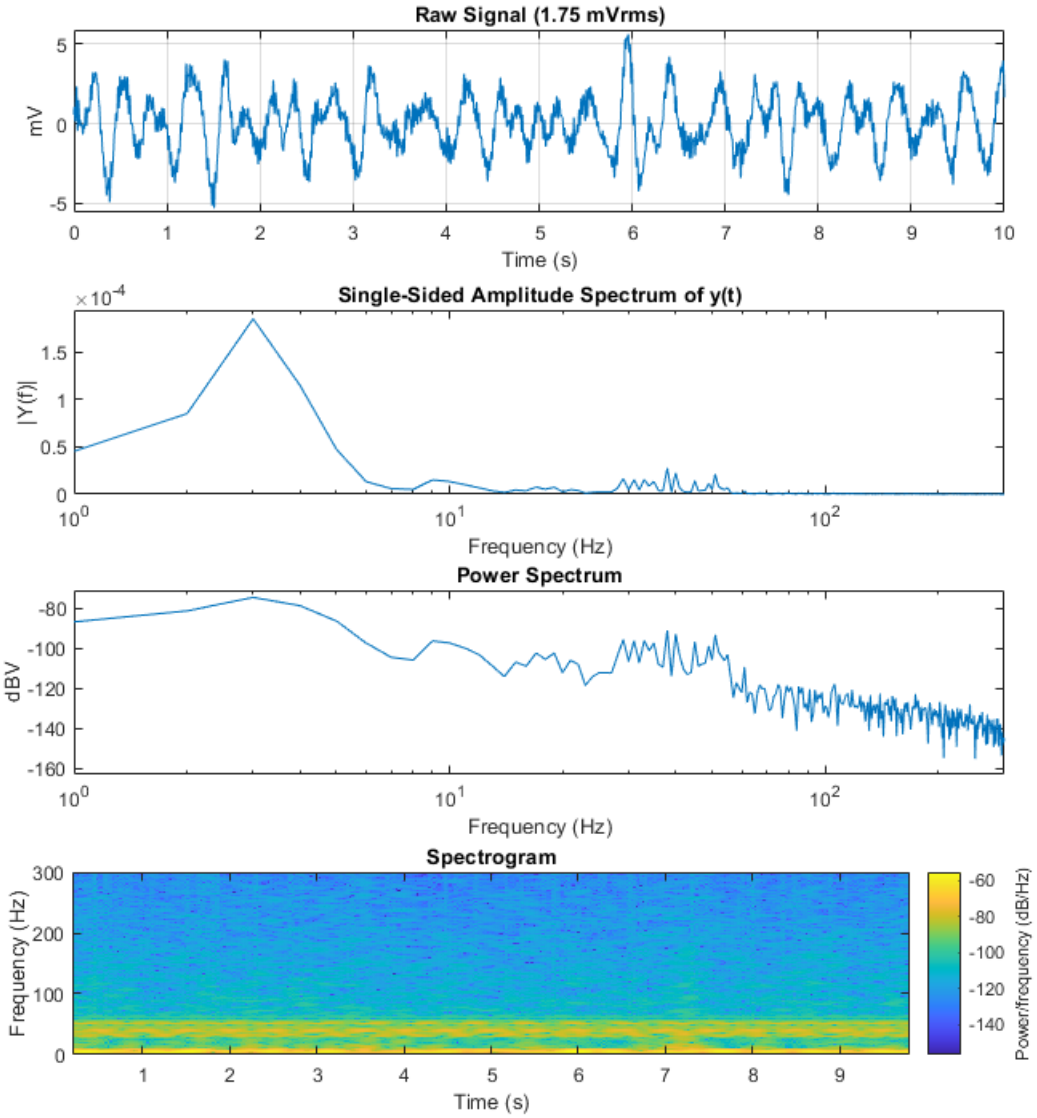
```
read from t=0.00s to t=10.00s
```

And that's it! Your data is now in MATLAB. The rest of the code is a plotting example.

LFP Analysis

Let's create a power spectrum and spectrogram plot of our data in the 0-300 Hz range.

```
TDTfft(data.streams.(STORE), 1, 'SPECPLOT', 1, 'FREQ', [0 300]);

% Make the figure larger
set(gcf,'Position',[100 100 750 800])
```


Raw Signal (1.75 mVrms)
mV
Time (s)
Single-Sided Amplitude Spectrum of y(t)
|Y(f)|
Frequency (Hz)
Power Spectrum
dBV
Frequency (Hz)
Spectrogram
Frequency (Hz)
Time (s)
Power/frequency (dB/Hz)

Note Filter Example

Import streaming EEG data into MATLAB using TDTbin2mat
Filter around behavioral events that were timestamped by the user
using the Run-time Notes feature in Synapse
Plot each occurrence in a subplot organized by Note type
Good for sleep scoring and behavioral discrimination

Download M File

Housekeeping

Clear workspace and close existing figures. Add SDK directories to MATLAB path.

```
close all; clear all; clc;
[MAINEXAMPLEPATH,name,ext] = fileparts(cd); % \TDTMatlabSDK\Examples
DATAPATH = fullfile(MAINEXAMPLEPATH, 'ExampleData'); %
\TDTMatlabSDK\Examples\ExampleData
[SDKPATH,name,ext] = fileparts(MAINEXAMPLEPATH); % \TDTMatlabSDK
addpath(genpath(SDKPATH));
```

Importing the Data

This example assumes you downloaded our example data sets and extracted it into the \TDTMatlabSDK\Examples directory. To import your own data, replace 'BLOCKPATH' with the path to your own data block.

In Synapse, you can find the block path in the database. Go to Menu → History. Find your block, then Right-Click → Copy path to clipboard.

```
BLOCKPATH = fullfile(DATAPATH,'Subject1-180426-120951');
```

Set up the variables for the data you want to extract. We will extract channel 1 from the EEG1 stream store.

```
STORE = 'EEG1';
CHANNEL = 1;
ONSET = -3; % relative onset, in seconds, from the note timestamp
```

Now read the specified data from our block into a MATLAB structure.

```
data = TDTbin2mat(BLOCKPATH, 'CHANNEL', CHANNEL);
```

```
Found Synapse note file: C:
\TDT\TDTMatlabSDK\Examples\ExampleData\Subject1-180426-120951\Notes.txt
read from t=0.00s to t=31.81s
```

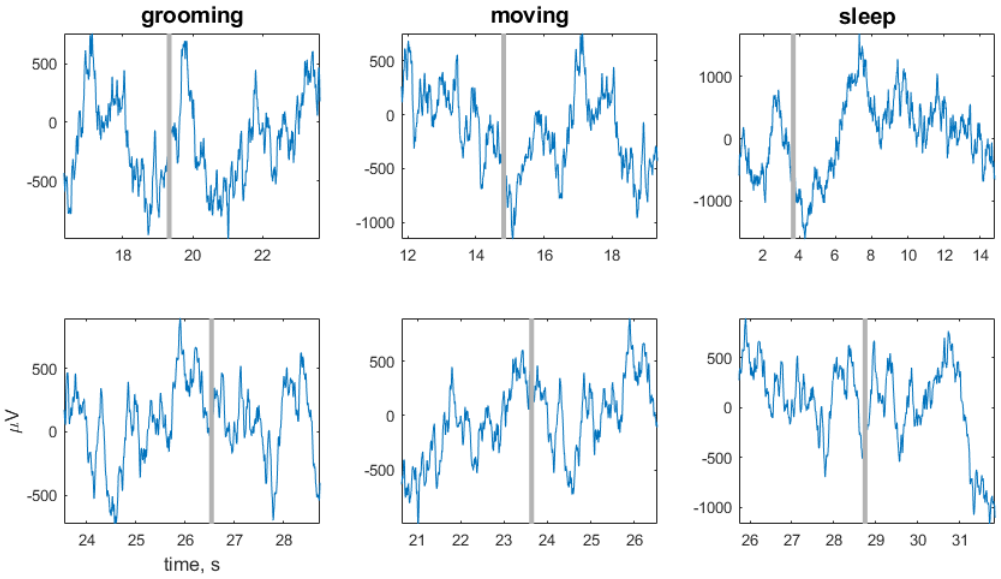
All notes are stored in a special epoc event called 'Note'

```
% find all the unique note values
notes = unique(data.epocs.Note.notes);

% find the highest number of occurrences (to inform our plot)
maxOccur = max(cell2mat(cellfun(@(x)
sum(ismember(data.epocs.Note.notes,x)),notes,'un',0)));
```

Loop through the notes for plotting

```
h = figure;
for i = 1:numel(notes)

    fprintf('Reading note: %s\n', notes{i})

    % look at only the data around this note type
    filtered = TDTfilter(data, 'Note', 'VALUES', notes{i}, 'TIME', ONSET);

    % for each note occurrence, plot the data from the note onset to the next
note onset
    n = numel(filtered.streams.(STORE).filtered);
    for j = 1:n
        plotInd = (j-1)*numel(notes)+i;
        subplot(maxOccur,numel(notes),plotInd);

        % x-axis is the valid time ranges, in seconds
        ts = filtered.time_ranges(1,j) + double(1:size(filtered.streams.
(STORE).filtered{j},2))/filtered.streams.(STORE).fs;

        % plot the snippet, in microvolts
        plot(ts, 1e6*filtered.streams.(STORE).filtered{j}'); hold on;

        % if we specified an ONSET, draw the vertical line at the note onset.
        if ONSET ~= 0
            line((ts(1)-ONSET)*[1 1], [min(1e6*filtered.streams.
(STORE).filtered{j}') max(1e6*filtered.streams.(STORE).filtered{j}')],
'Color',[.7 .7 .7], 'LineStyle','-', 'LineWidth', 3);
        end

        % plot labels
        if j == 1
            title(notes{i},'FontSize',14)
        elseif j == n
            if i == 1
                ylabel('\muV','FontSize',12)
                xlabel('time, s','FontSize',12)
            end
        end
        axis tight;
    end
end

set(gcf, 'Position',[100, 100, 1000, 500])
```

```
Reading note: grooming
Reading note: moving
Reading note: sleep
```



Raster Peristimulus Time Histogram (PSTH) Example

Import snippet and epoc data into MATLAB using TDTbin2mat
Generate peristimulus raster and histogram plots over all trials
Good for stim-response experiments, such as optogenetic or electrical stimulation

Download M File

Housekeeping

Clear workspace and close existing figures. Add SDK directories to MATLAB path.

```
close all; clear all; clc;
[MAINEXAMPLEPATH,name,ext] = fileparts(cd); % \TDTMatlabSDK\Examples
DATAPATH = fullfile(MAINEXAMPLEPATH, 'ExampleData'); %
\TDTMatlabSDK\Examples\ExampleData
[SDKPATH,name,ext] = fileparts(MAINEXAMPLEPATH); % \TDTMatlabSDK
addpath(genpath(SDKPATH));
```

Importing the Data

This example assumes you downloaded our example data sets and extracted it into the \TDTMatlabSDK\Examples directory. To import your own data, replace 'BLOCKPATH' with the path to your own data block.

In Synapse, you can find the block path in the database. Go to Menu → History. Find your block, then Right-Click → Copy path to clipboard.

```
BLOCKPATH = fullfile(DATAPATH,'Algernon-180308-130351');
```

Set up the variables for the data you want to extract. We will extract channel 1 from the eNe1 snippet data store, created by the PCA Sorting gizmo, and use our PulseGen epoc event ('PC0/') as our stimulus onset.

```
REF_EPOC = 'PC0/';
SNIP_STORE = 'eNe1';
SORTID = 'TankSort';
CHANNEL = 3;
SORTCODE = 0; % set to 0 to use all sorts
TRANGE = [-0.3, 0.8]; % window size [start time relative to epoc onset,
window duration]
```

Now read the specified data from our block into a MATLAB structure. The 'NODATA' flag means that we are only interested in the snippet timestamps, not the actual snippet waveforms in this example.

```
data = TDTbin2mat(BLOCKPATH, 'TYPE', {'epocs', 'snips', 'scalars'},
'SORTNAME', SORTID, 'CHANNEL', CHANNEL, 'NODATA', 1);
```

```
read from t=0.00s to t=61.23s
```

Use TDTfilter to extract data around our epoc event

Using the 'TIME' parameter extracts data only from the time range around our epoc event.

```
raster_data = TDTfilter(data, REF_EPOC, 'TIME', TRANGE);
```

Adding the 'TIMEREF' flag makes all of the timestamps relative to the epoc event, which is ideal for generating histograms.

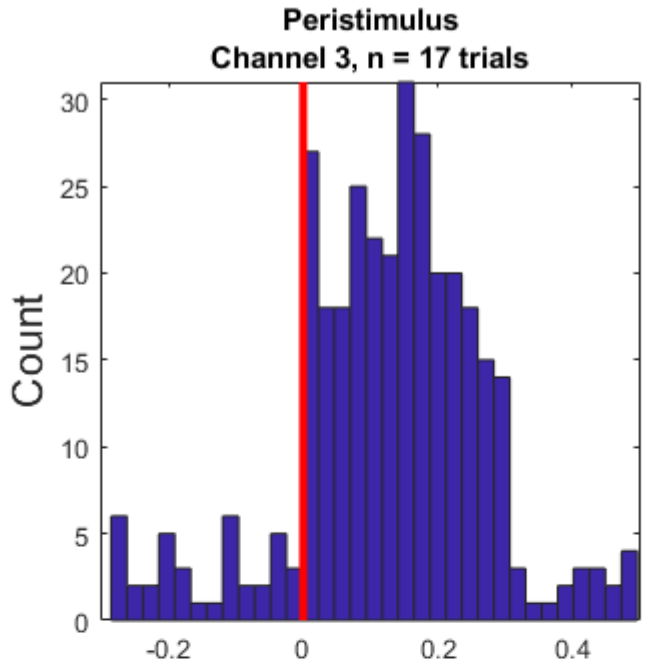
```
hist_data = TDTfilter(data, REF_EPOC, 'TIME', TRANGE, 'TIMEREF', 1);
```

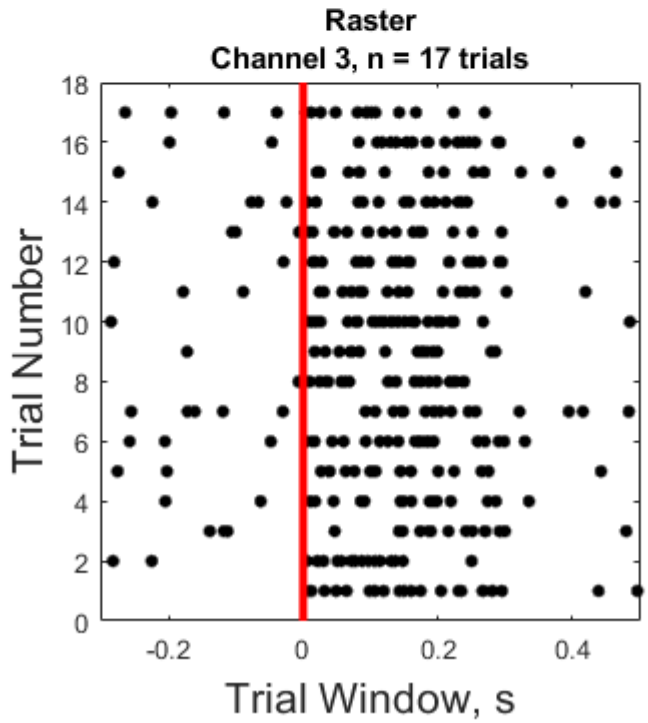
And that's it! Your data is now in MATLAB. The rest of the code is a simple plotting example. First, we'll find matching timestamps for our selected sort code (unit).

```
TS = raster_data.snips.(SNIP_STORE).ts;
if SORTCODE ~= 0
    i = find(raster_data.snips.(SNIP_STORE).sortcode == SORTCODE);
    TS = TS(i);
end
if isempty(TS)
    error('no matching timestamps found')
end

num_trials = size(raster_data.time_ranges, 2);
```

Make the histogram plot

```
figure('Position',[100, 100, 500, 800]);

hist_TS = hist_data.snips.(SNIP_STORE).ts;
subplot(2,1,1);
NBINS = floor(numel(hist_TS)/10);
hist(hist_TS, NBINS);
N = hist(hist_TS, NBINS); hold on;

axis tight; axis square;
set(gca, 'XLim', [TRANGE(1), TRANGE(1)+TRANGE(2)]);
ylabel('Count','FontSize',16);
title({'Peristimulus', sprintf('Channel %d, n = %d trials', CHANNEL,
num_trials)});

% Draw a vertical line at t=0.
line([0 0], [0, max(N)], 'Color','r', 'LineStyle','-', 'LineWidth', 3);

% Creating the Raster Plot
% For the raster plot, make a cell array of timestamps for each trial.
all_TS = cell(num_trials, 1);
all_Y = cell(num_trials, 1);
for trial = 1:num_trials
    trial_on = raster_data.time_ranges(1, trial);
    trial_off = raster_data.time_ranges(2, trial);
    trial_TS = TS(TS >= trial_on & TS < trial_off);
    all_TS{trial} = trial_TS - trial_on + TRANGE(1);
    all_Y{trial} = trial * ones(numel(trial_TS), 1);
end
all_X = cat(1, all_TS{:});
all_Y = cat(1, all_Y{:});

% Make the raster plot.
subplot(2,1,2);
plot(all_X, all_Y, '.', 'MarkerEdgeColor','k', 'MarkerSize',15); hold on;
axis tight; axis square;
set(gca, 'XLim', [TRANGE(1), TRANGE(1)+TRANGE(2)]);
xlabel('Trial Window, s','FontSize',16);
ylabel('Trial Number','FontSize',16);
title({'Raster', sprintf('Channel %d, n = %d trials', CHANNEL, num_trials)});

% Draw a vertical line at t=0.
line([0 0], [0, trial+1], 'Color','r', 'LineStyle','-', 'LineWidth', 3);
```


Peristimulus
Channel 3, n = 17 trials
Count
30
25
20
15
10
5
0
-0.2
0
0.2
0.4


Raster
Channel 3, n = 17 trials
Trial Number
18
16
14
12
10
8
6
4
2
0
-0.2
0
0.2
0.4
Trial Window, s

Filter and Threshold Raw Data Into Snippets

Import raw streaming data into MATLAB using TDTbin2mat
Digitally filter the single unit data using TDTdigitalfilter
Threshold and extract snippets using TDTthresh

Download M File

Housekeeping

Clear workspace and close existing figures. Add SDK directories to MATLAB path.

```
close all; clear all; clc;
[MAINEXAMPLEPATH,name,ext] = fileparts(cd); % \TDTMatlabSDK\Examples
DATAPATH = fullfile(MAINEXAMPLEPATH, 'ExampleData'); %
\TDTMatlabSDK\Examples\ExampleData
[SDKPATH,name,ext] = fileparts(MAINEXAMPLEPATH); % \TDTMatlabSDK
addpath(genpath(SDKPATH));
```

Importing the Data

This example assumes you downloaded our example data sets and extracted it into the \TDTMatlabSDK\Examples directory. To import your own data, replace 'BLOCKPATH' with the path to your own data block.

In Synapse, you can find the block path in the database. Go to Menu → History. Find your block, then Right-Click → Copy path to clipboard.

```
BLOCKPATH = fullfile(DATAPATH,'Algernon-180308-130351');
```

Set up the variables for the data you want to extract. We will extract channel 1 from the Wav1 data store, created by the Stream Data Storage gizmo, and apply our threshold at -125 uV to create 30 sample snippets.

```
STORE = 'Wav1';
CHANNEL = 1;
THRESH = -125e-6; % define snippet when it crosses this threshold
NPTS = 30; % size of the snippet
OVERLAP = 0; % set to 1 to allow double crossings within same window to count
as a different snippet
```

Now read the specified data from our block into a MATLAB structure.

```
data = TDTbin2mat(BLOCKPATH, 'STORE', STORE, 'CHANNEL', CHANNEL);
```

```
read from t=0.00s to t=61.23s
```

Use TDTdigitalfilter to filter the streaming waveforms

We are interested in single unit activity in the 300-5000 Hz band.

```
su = TDTdigitalfilter(data, STORE, [300 5000]);
```

Use TDTthresh to extract snippet events

Mode is set to 'manual' in this example for extracting snippets around a hard threshold.

```
su = TDTthresh(su, STORE, 'MODE', 'manual', 'THRESH', THRESH, 'NPTS', NPTS,
'OVERLAP', OVERLAP);

% You could also use 'auto' detection, which sets a moving threshold base
% on a multiple of the RMS of a previous time period. For example,
% set the threshold at 6*RMS of the previous 5 seconds of data
%su = TDTthresh(su, STORE, 'MODE', 'auto', 'POLARITY', -1, 'STD', 6, 'TAU',
5);
```

```
window size is 30, without overlap.
using absolute threshold method, set to -125.00 uV.
```

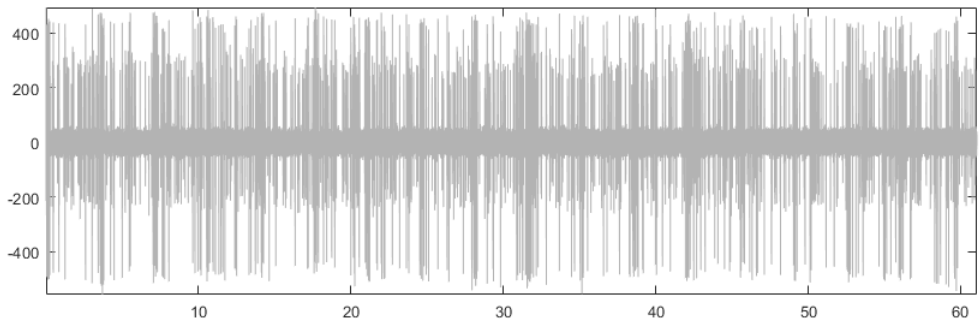
And that's it! Your data is now in MATLAB and thresholded. TDTthresh adds a new store into the data structure called data.snips.Snip that we will now use for plotting. First, we'll get some properties of the snippets that we'll use later for scatter plots

```
maxvals = max(su.snips.Snip.data, [], 2)*1e6;
minvals = min(su.snips.Snip.data, [], 2)*1e6;
```

Begin plotting

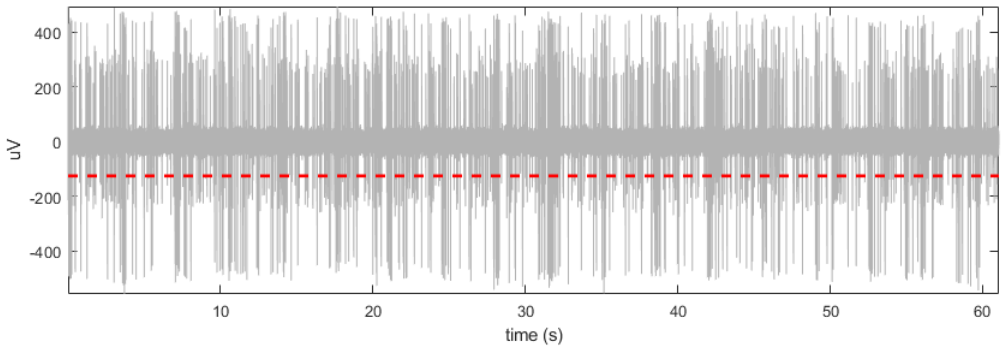
Plot the single unit data with snippet overlay.

```
subplot(2,3,[1 2 3])
ts = (1:numel(su.streams.(STORE).data(1,:)))/su.streams.(STORE).fs;
plot(ts, su.streams.(STORE).data(1,:)*1e6, 'color', [.7 .7 .7]);
axis([ts(1) ts(end) min(minvals) max(maxvals)]); ax1 = axis();
hold on;

% Position the figure
set(gcf, 'Position', [100 100 1000 700])
```



Plot the threshold line.

```
if numel(su.snips.Snip.thresh) == 1
    thresh = su.snips.Snip.thresh*ones(size(ts));
else
    thresh = su.snips.Snip.thresh;
end
plot(ts, thresh*1e6, 'r--', 'LineWidth', 2)
ylabel('uV');
xlabel('time (s)');
```

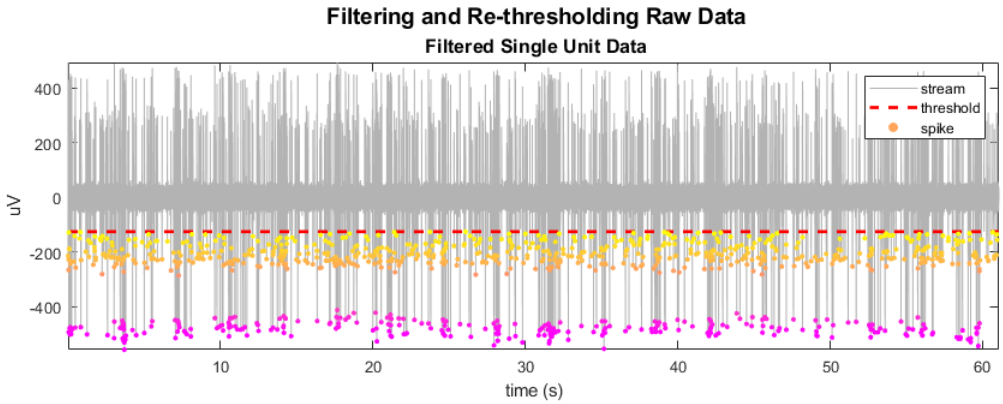


Define a color map based on the minimum values. The dots are not colored by any complex sorting algorithm, just by magnitude. We'll use this coloring to track the snippets through the rest of the plots. This uses the vals2colormap function, which can be found at: <https://github.com/vistalab/vistateach/blob/master/cogneuro/tutorial1_timeseries/vals2colormap.m>.

```
colors = vals2colormap(minvals, 'spring', [min(minvals)*.9 max(minvals)]);
```

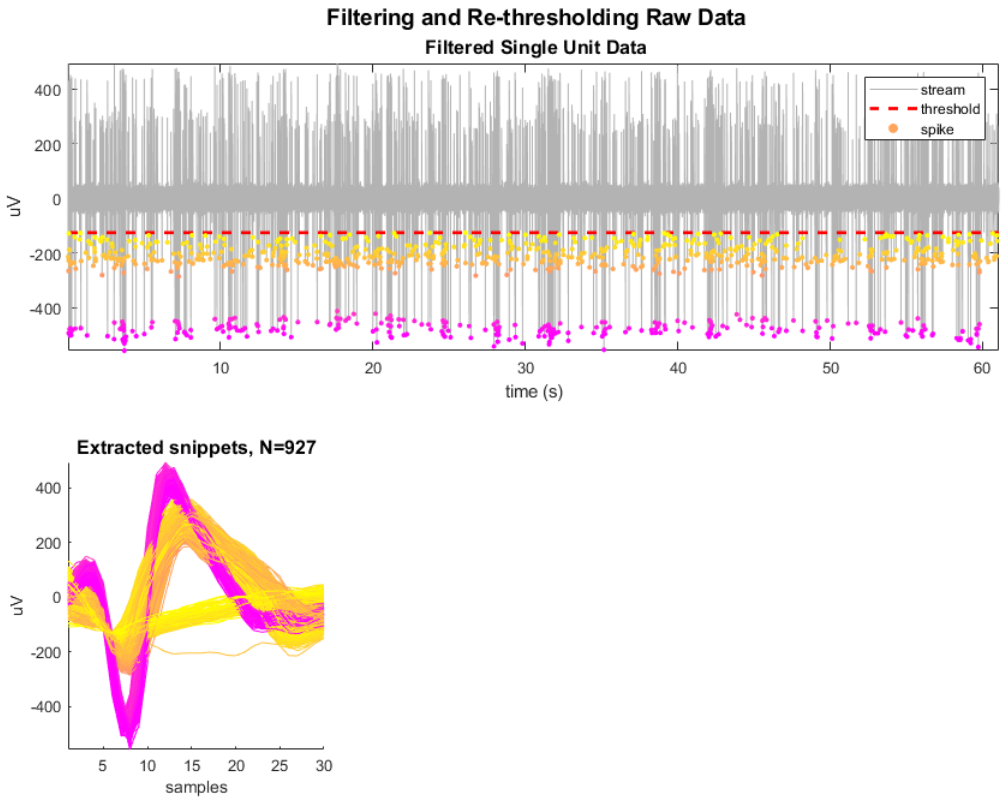
Overlay the min values as dots on the streaming plot.

```
scatter(su.snips.Snip.ts, minvals, 10, colors, 'filled');

legend({'stream', 'threshold', 'spike'});
% add a fancy latex title
title(['\fontsize{14} Filtering and Re-thresholding Raw Data' char(10) ...
    '\fontsize{12} Filtered Single Unit Data'],'interpreter','tex');

% Alternatively, overlay entire snippets on time window.
%npts = size(su.snips.Snip.data,2);
%for ii = 1:size(su.snips.Snip.data, 1)
%    plot(su.snips.Snip.ts(ii)+((1:npts)-floor(npts/4))/su.snips.Snip.fs,
su.snips.Snip.data(ii,:)*1e6, 'color', colors(ii,:));
%end
```



Plot the extracted snippets.

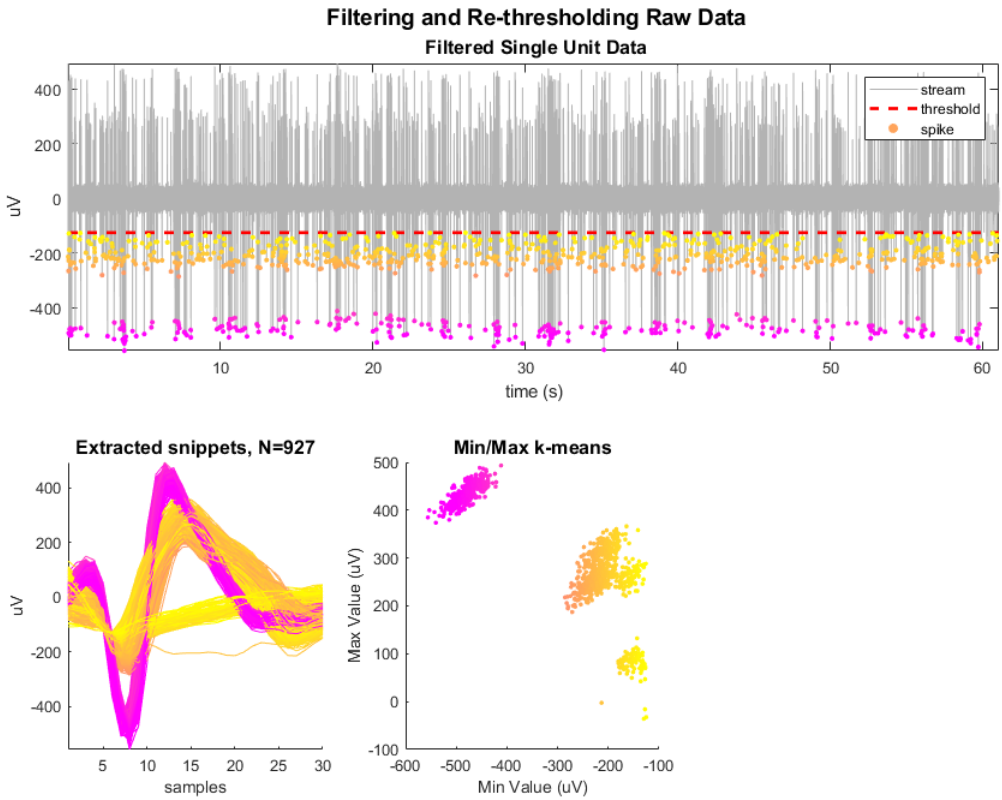
```
subplot(2,3,4)
hold on;
for ii = 1:size(su.snips.Snip.data, 1)
    plot(su.snips.Snip.data(ii,:)*1e6, 'color', colors(ii,:));
end
axis tight;
xlabel('samples')
ylabel('uV')
title(sprintf('Extracted snippets, N=%d', ii), 'FontSize', 12)
```



Do Min/Max k-Means and PCA analysis

Plot snippets in a simple feature space.

```
subplot(2,3,5)
scatter(minvals, maxvals, 8, colors, 'filled')
xlabel('Min Value (uV)')
ylabel('Max Value (uV)')
title('Min/Max k-means', 'FontSize', 12)
hold on;
```

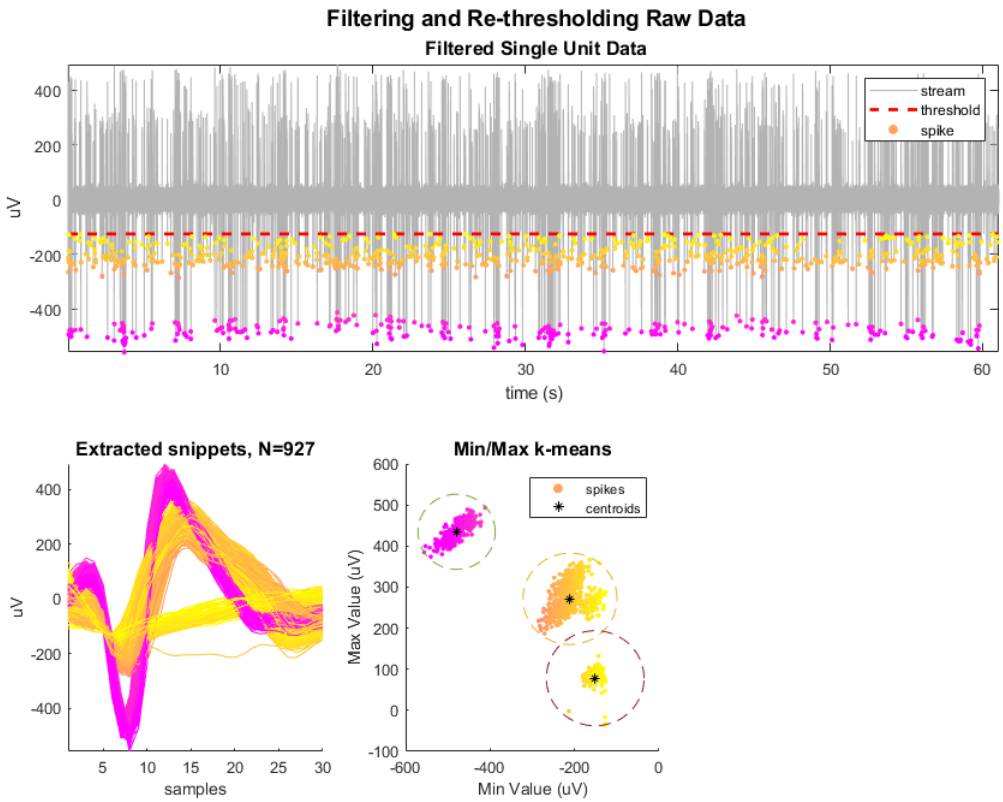


Do k-means for 3 clusters.

```
rand('seed', 1); % for reproducibility
[idx, C] = kmeans([minvals, maxvals], 3);
```

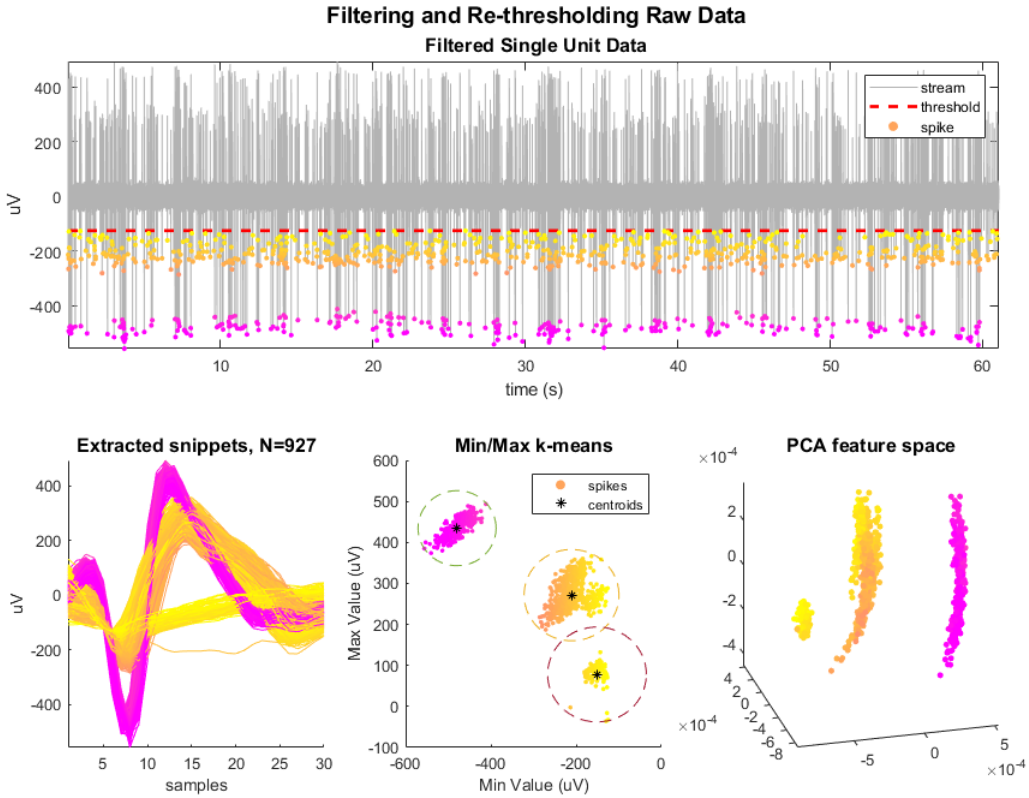
Plot centroids and cluster circles

```
for ii = 1:max(idx)

    % plot centroid
    scatter(C(ii,1), C(ii,2), 30, '*k');

    % find spike with max distance from center to use as radius for circle
    x_center = C(ii, 1);
    y_center = C(ii, 2);
    curr_mins = minvals(idx==ii);
    curr_maxs = maxvals(idx==ii);
    dist = sqrt((curr_mins - x_center).^2 + (curr_maxs - y_center).^2);
    r = max(dist);

    % plot circle
    th = 0:pi/50:2*pi;
    xunit = r * cos(th) + x_center;
    yunit = r * sin(th) + y_center;
    plot(xunit, yunit, '--');
end
legend({'spikes','centroids'})
```



Plot snippets in PCA space.

```
subplot(2,3,6)
hold on;
try
    [coeff,score,latent] = pca(su.snips.Snip.data);  % newer MATLAB uses this
catch
    [coeff,score,latent] = princomp(su.snips.Snip.data);
end

[idx3, C3] = kmeans(score(:,1:3), 3);
for ii = 1:max(idx3)

    % apply sort codes to our data
    su.snips.Snip.sortcode(idx3==ii) = ii;

    % get principle components and color for this cluster
    pca1 = score(idx3==ii,1);
    pca2 = score(idx3==ii,2);
    pca3 = score(idx3==ii,3);
    color = colors(idx3==ii,:);

    % plot dots
    scatter3(pca1, pca2, pca3, 15, color, 'filled')

    % plot centroid
    %scatter3(C3(ii,1), C3(ii,2), C3(ii,3), 75, 'k', 'filled');
end
axis tight;
view([-15 25]); % set initial view on 3-D plot
title('PCA feature space', 'FontSize', 12)

% keep sort code 1 only
idx = su.snips.Snip.sortcode == 1;
su.snips.Snip.sortcode(~idx) = [];
su.snips.Snip.ts(~idx) = [];
```


Filtering and Re-thresholding Raw Data
Filtered Single Unit Data
stream
threshold
spike
uV
time (s)
Extracted snippets, N=927
samples
Min/Max k-means
spikes
centroids
Max Value (uV)
Min Value (uV)
PCA feature space

Snippet Plot Example

Import snippet data into MATLAB using TDTbin2mat
Sort snippets based on channel number and sort code for any number of channels and sort codes
Plot the average waveform shape and standard deviation for 16 channels and three sort codes
Good for spike sorting and first-pass visualization of sorted waveforms

Download M File

Housekeeping

Clear workspace and close existing figures. Add SDK directories to MATLAB path.

```
close all; clear all; clc;
[MAINEXAMPLEPATH,name,ext] = fileparts(cd); % \TDTMatlabSDK\Examples
DATAPATH = fullfile(MAINEXAMPLEPATH, 'ExampleData'); %
\TDTMatlabSDK\Examples\ExampleData
[SDKPATH,name,ext] = fileparts(MAINEXAMPLEPATH); % \TDTMatlabSDK
addpath(genpath(SDKPATH));
```

Importing the Data

This example assumes you downloaded our example data sets and extracted it into the \TDTMatlabSDK\Examples directory. To import your own data, replace 'BLOCKPATH' with the path to your own data block.

In Synapse, you can find the block path in the database. Go to Menu → History. Find your block, then Right-Click → Copy path to clipboard.

```
BLOCKPATH = fullfile(DATAPATH,'Algernon-180308-130351');
```

Now read snippet data into a MATLAB structure called 'data'.

```
data = TDTbin2mat(BLOCKPATH, 'TYPE', {'snips'});
```

```
read from t=0.00s to t=61.23s
```

And that's it! Your data is now in MATLAB. The rest of the code describes sorting snippets based on channel number and sort code, then plotting a subselection of three sort codes.

Spike Snippet Sorting

Collect waveforms, averages, standard deviation, and snippet times of all snippets, sorted by channel and sortcode number.

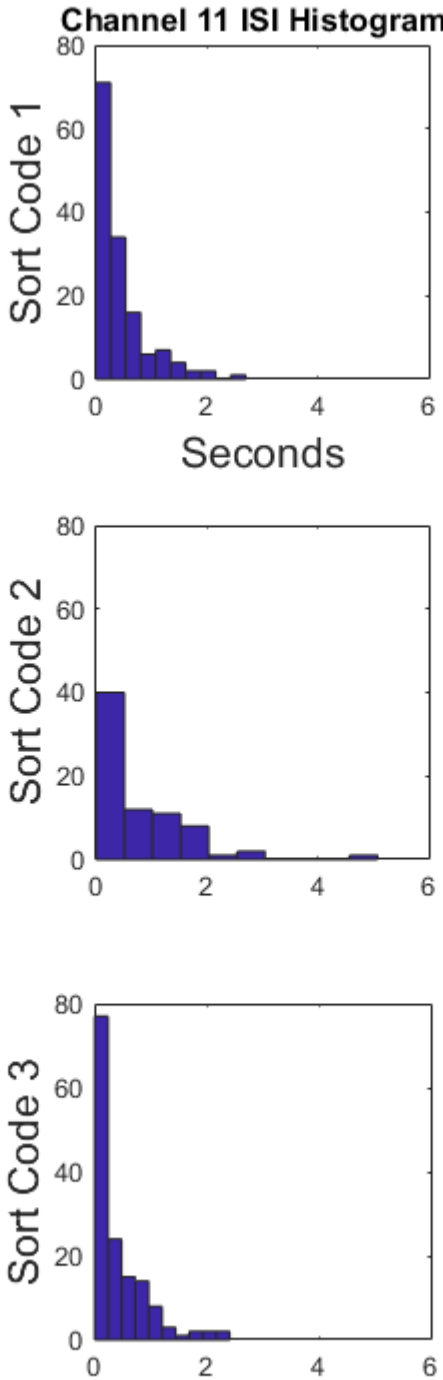
```
% Note: If you want a pile plot of *ALL* snippets (every channel), use this:
% figure;
% plot(samples, data.snips.eNe1.data(:,:));

% Pull field name as string from snip store
SNIP_STORE = fieldnames(data.snips);
SNIP_STORE = SNIP_STORE{1};

% Useful variables
numchans = single(max(data.snips.(SNIP_STORE).chan));
if numchans > 64
    warning('This display function is only good for 64 channels or fewer.
Truncating to 64 channels');
    numchans = 64;
end

nsamples = length(data.snips.(SNIP_STORE).data(1,:));
sorts = sort(unique(data.snips.(SNIP_STORE).sortcode))';

% Remove unsorted and outliers from analysis unless it's the only one
if numel(sorts) ~= 1
    sorts(sorts==0 | sorts==31) = [];
end

% Declare stores for sort codes, averages, standard deviations, and timestamps
sorted_stores = cell(numchans, numel(sorts));
store_averages = cell(numchans, numel(sorts));
sorted_stdp = cell(numchans, numel(sorts));
sorted_stdn = cell(numchans, numel(sorts));
snip_times = cell(numchans, numel(sorts));
snip_isi = cell(numchans, numel(sorts));

% Filter based on channel and sort code
for chan = 1:numchans
    for sort_ind = 1:numel(sorts)
        sort_code = sorts(sort_ind);
        % Create index for data that matches current channel and sortcode
        i = find(data.snips.(SNIP_STORE).chan == chan & data.snips.
(SNIP_STORE).sortcode == sort_code);
        sorted_stores{chan, sort_code} = 1e6*data.snips.
(SNIP_STORE).data(i,:); % scaled to uV
        store_averages{chan, sort_code} = sum(sorted_stores{chan, sort_code}/
length(sorted_stores{chan, sort_code}));
        if size(sorted_stores{chan, sort_code}, 1) > 1
            sorted_stdp{chan, sort_code} = std(sorted_stores{chan, sort_code})
+ store_averages{chan, sort_code};
            sorted_stdn{chan, sort_code} = store_averages{chan, sort_code} -
std(sorted_stores{chan, sort_code});
        else
            sorted_stdp{chan, sort_code} = zeros(1, size(sorted_stores{chan,
```

```
sort_code},2));
            sorted_stdn{chan, sort_code} = zeros(1, size(sorted_stores{chan,
sort_code},2));
        end
        % Find timestamps of snips
        snip_times{chan, sort_code} = data.snips.(SNIP_STORE).ts(i);

        % Inter-spike Interval of sorted snips
        snip_isi{chan,sort_code} = diff(snip_times{chan,sort_code});
    end
end

% Use this code block to extract unsorted and outlier snips
% unsorted = cell(numchans,1);
% outliers = cell(numchans,1);
%
% for CHANNEL = 1:numchans
%     i = data.snips.(SNIP_STORE).chan == CHANNEL & data.snips.
(SNIP_STORE).sortcode == 0;
%     unsorted{CHANNEL, 1} = data.snips.(SNIP_STORE).data(i,:);
%
%     j = data.snips.(SNIP_STORE).chan == CHANNEL & data.snips.
(SNIP_STORE).sortcode == 31;
%     outliers{CHANNEL, 1} = data.snips.(SNIP_STORE).data(j,:);
% end
```

Generate an ISI histogram for selected channels

Look at ISI histogram for first 3 sort codes

```
PLOT_CHANS = [11]; % change to 1:numchans to plot all of them

ax = cell(1,3);
for chan = PLOT_CHANS
    figure('Name','ISI Histograms','Position',[900, 100, 500, 800]);
    for sort_ind = 1:size(snip_isi,2)
        % skip sort code if there are none
        if size(snip_isi{chan, sort_ind},1) == 0, continue, end

        % make histogram on new subplot
        ax{sort_ind} = subplot(3,1,sort_ind);
        hist(snip_isi{chan,sort_ind});

        if sort_ind == 1
            title(sprintf('Channel %d ISI Histograms',chan),'FontSize',12)
            xlabel('Seconds','FontSize',16)
        end
        ylabel(sprintf('Sort Code %d',sorts(sort_ind)),'FontSize',16)
        axis square
    end
    linkaxes([ax{:}],'xy');
end
```

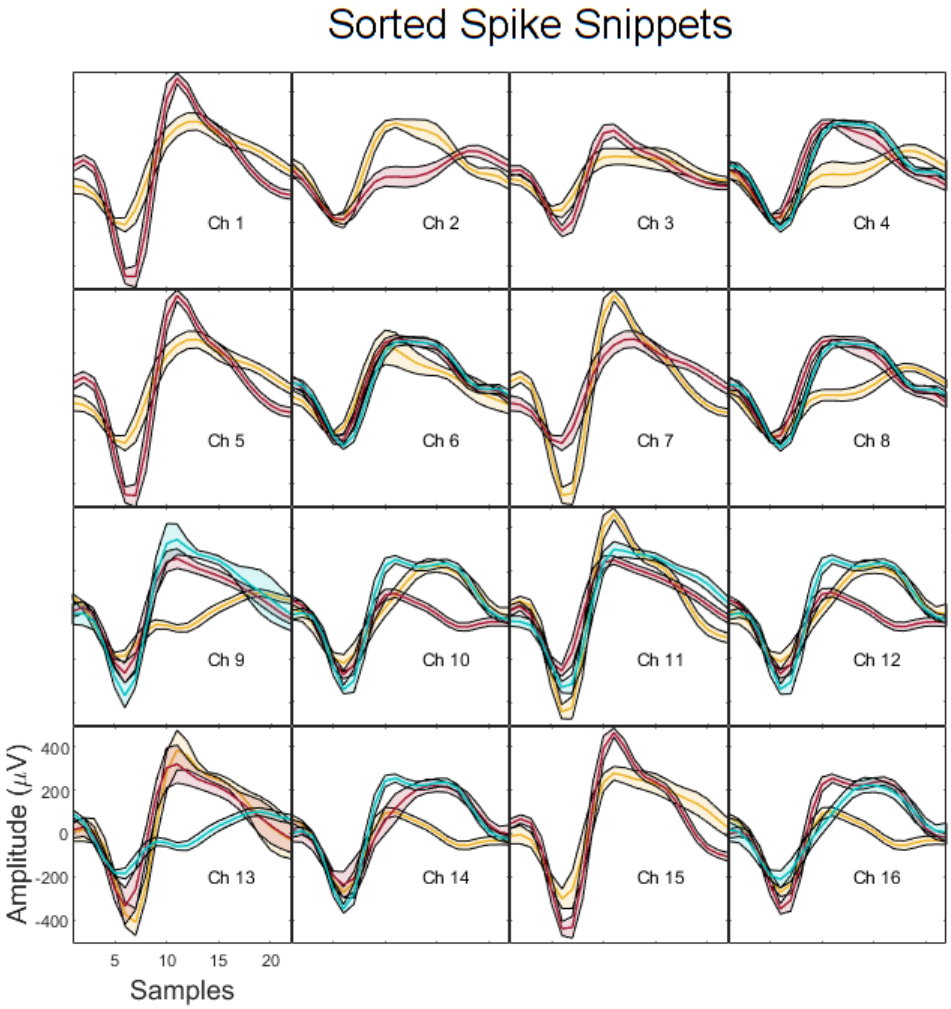

Channel 11 ISI Histograms
Sort Code 1
Sort Code 2
Sort Code 3
Seconds
0
20
40
60
80
0
2
4
6

Waveform Plots

Create filled waveform plots with channels and sortcodes
Waveforms will be the average waveform of all snippets for each sortcode for each channel, with a standard deviation fill around it
Note: The plot below uses up to 64 channels and 3 sort codes but can be modified

```
% Samples array for x-axis of fill plots
sample_arr = 1:nsamples;
XX = [sample_arr, fliplr(sample_arr)];

% set plot locations based on channel count
numcols = double(ceil(sqrt(numchans)));
numrows = double(ceil(numchans/numcols));
spc = sqrt(numrows*numcols)+1.5;
indt1 = 1/(numrows*numrows);
indt2 = 0.8-indt1;

h = figure('Name','Sorted Spike Snippets','Position', [100, 100, 800, 800]);

% Add master title to subplots using figure uicontrol
uicontrol('Style','text','String','Sorted Spike Snippets',...
    'FontSize',20','HorizontalAlignment','center','Units','normalized',...
    'Position', [0 .93 1 .05],'BackgroundColor',[1 1 1]);

% Default colors for sort codes 1, 2, 3
colors = {[0.9290, 0.6940, 0.1250],[0.6350, 0.0780, 0.1840],[0, 0.75, 0.75]};

max_axis = zeros(1,4);
ax = zeros(1, numchans);

row = 1;
col = 1;
last_sort = min(size(snip_isi,2),3);
for chan = 1:numchans
    for sort_ind = 1:last_sort
        % Used for filling Std Dev in with Fill later
        YY = [sorted_stdp{chan,sort_ind}, fliplr(sorted_stdn{chan,sort_ind})];

        pos = [col/spc-indt1 indt2-(row-1)/spc 1/spc 1/spc];
        ax(chan) = subplot('Position',pos);

        % create an empty axis if no sort codes found
        if all(cell2mat(store_averages(chan,1:last_sort)) == 0) && sort_ind ==
last_sort
            plot(zeros(size(store_averages{chan,sort_ind})),'w','LineWidth',
1);
        else
            if ~all(store_averages{chan,sort_ind} == 0)
                % Std dev fill around averaged waveform
                h = fill(XX, YY, colors{sort_ind});
                set(h, 'facealpha', .15); %set transparency of fill plot
                hold on;

plot(store_averages{chan,sort_ind},'color',colors{sort_ind},'LineWidth',1);
            end
        end
```

```
        if sort_ind == last_sort
            % Channel labels
            text(15-sqrt(col), -200, sprintf('Ch %d',chan), 'FontSize', 14/
((numrows^(1/3))));
            axis square;
            axis tight;
        end

        % Only add axis labels on left column and bottom row
        if row == numrows && col == 1
            xlabel('Samples','FontSize',16/((numrows^(1/8))));
            ylabel('Amplitude (\muV)','FontSize',16/((numrows^(1/8))));
        else
            % Get rid of the numbers but leave the ticks.
            set(ax(chan),'Xticklabel',[]);
            set(ax(chan),'Yticklabel',[]);
        end
    end

    % fill one row at a time, go to the next row when col == numcols
    if col == numcols
        row = row+1;
        col = 1;
    else
        col = col+1;
    end

end
% use same axes for all plots
linkaxes(ax, 'xy');
```


Sorted Spike Snippets
Ch 1
Ch 2
Ch 3
Ch 4
Ch 5
Ch 6
Ch 7
Ch 8
Ch 9
Ch 10
Ch 11
Ch 12
Ch 13
Ch 14
Ch 15
Ch 16
Amplitude (μV)
400
200
0
-200
-400
5
10
15
20
Samples

Stream Plot Example

Import continuous data into MATLAB using TDTbin2mat
Plot a single channel of data with various filtering schemes
Good for first-pass visualization of streamed data

Download M File

Housekeeping

Clear workspace and close existing figures. Add SDK directories to MATLAB path.

```
close all; clear all; clc;
[MAINEXAMPLEPATH,name,ext] = fileparts(cd); % \TDTMatlabSDK\Examples
DATAPATH = fullfile(MAINEXAMPLEPATH, 'ExampleData'); %
\TDTMatlabSDK\Examples\ExampleData
[SDKPATH,name,ext] = fileparts(MAINEXAMPLEPATH); % \TDTMatlabSDK
addpath(genpath(SDKPATH));
```

Importing the Data

This example assumes you downloaded our example data sets and extracted it into the \TDTMatlabSDK\Examples directory. To import your own data, replace 'BLOCKPATH' with the path to your own data block.

In Synapse, you can find the block path in the database. Go to Menu → History. Find your block, then Right-Click → Copy path to clipboard.

```
BLOCKPATH = fullfile(DATAPATH,'Algernon-180308-130351');
```

Now read channel 1 from all stream data into a MATLAB structure called 'data'.

```
data = TDTbin2mat(BLOCKPATH, 'TYPE', {'streams', 'epocs'}, 'CHANNEL', 1);
```

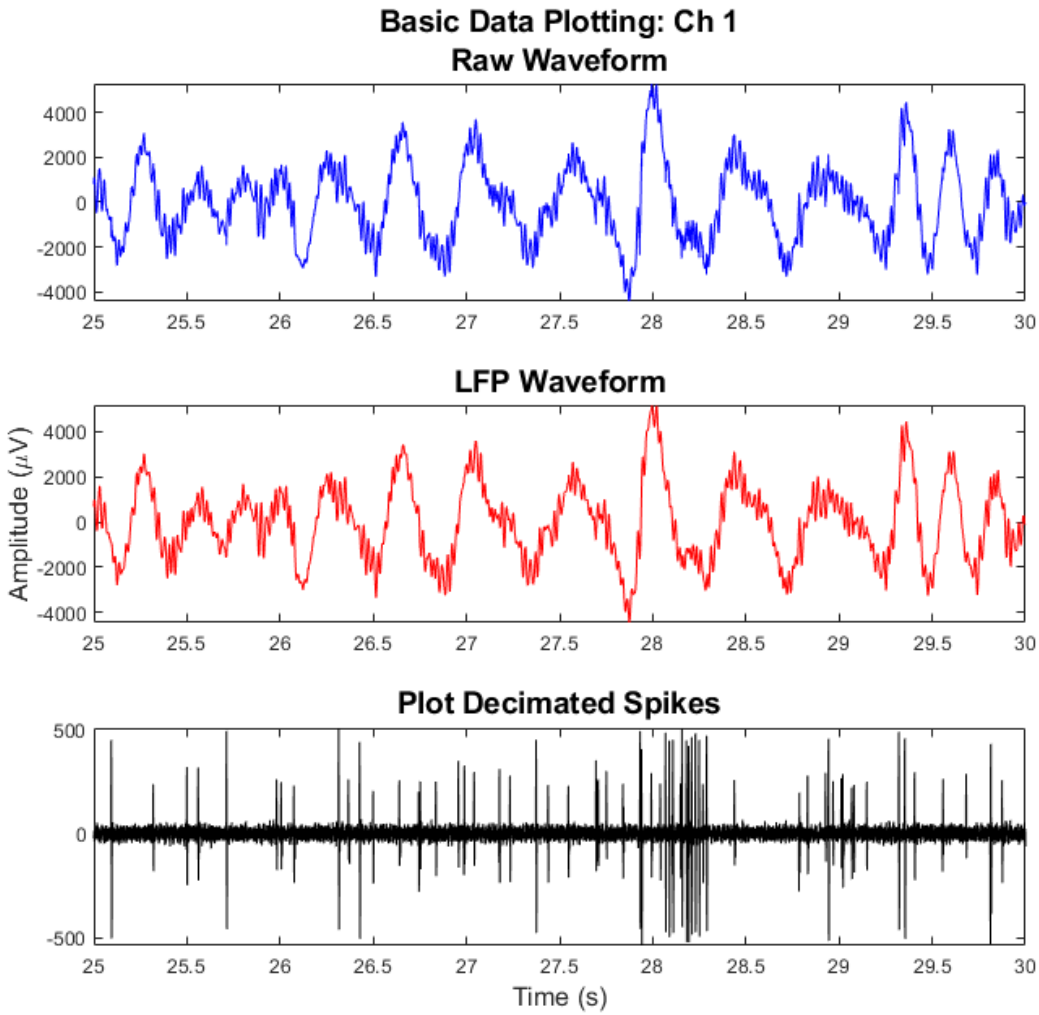
```
read from t=0.00s to t=61.23s
```

And that's it! Your data is now in MATLAB. The rest of the code is a simple plotting example.

Stream Store Plotting

Let's create time vectors for each stream store for plotting in time.

```
time_Wav1 = (1:length(data.streams.Wav1.data))/data.streams.Wav1.fs;
time_LFP1 = (1:length(data.streams.LFP1.data))/data.streams.LFP1.fs;
time_pNe1 = (1:length(data.streams.pNe1.data))/data.streams.pNe1.fs;

ax1 = subplot(3,1,1);
plot(time_Wav1, data.streams.Wav1.data(1,:)*1e6,'b');
axis tight;
title({'Basic Data Plotting: Ch 1','Raw Waveform'},'FontSize',14)
xlim([25 30]);

ax2 = subplot(3,1,2);
plot(time_LFP1, data.streams.LFP1.data(1,:)*1e6,'r');
title('LFP Waveform','FontSize',14);

ax3 = subplot(3,1,3);
plot(time_pNe1, data.streams.pNe1.data(1,:),'k');
title('Plot Decimated Spikes','FontSize',14);

ax = [ax1, ax2, ax3];
axis([ax1 ax2], 'tight')
linkaxes(ax, 'x')
xlim(ax(end), [25 30]);
xlabel(ax(end), 'Time (s)','FontSize',12)
ylabel(ax(2), 'Amplitude (\muV)','FontSize',12);

% Enlarge figure.
set(gcf, 'Units', 'centimeters', 'OuterPosition', [10, 10, 20, 20]);
```



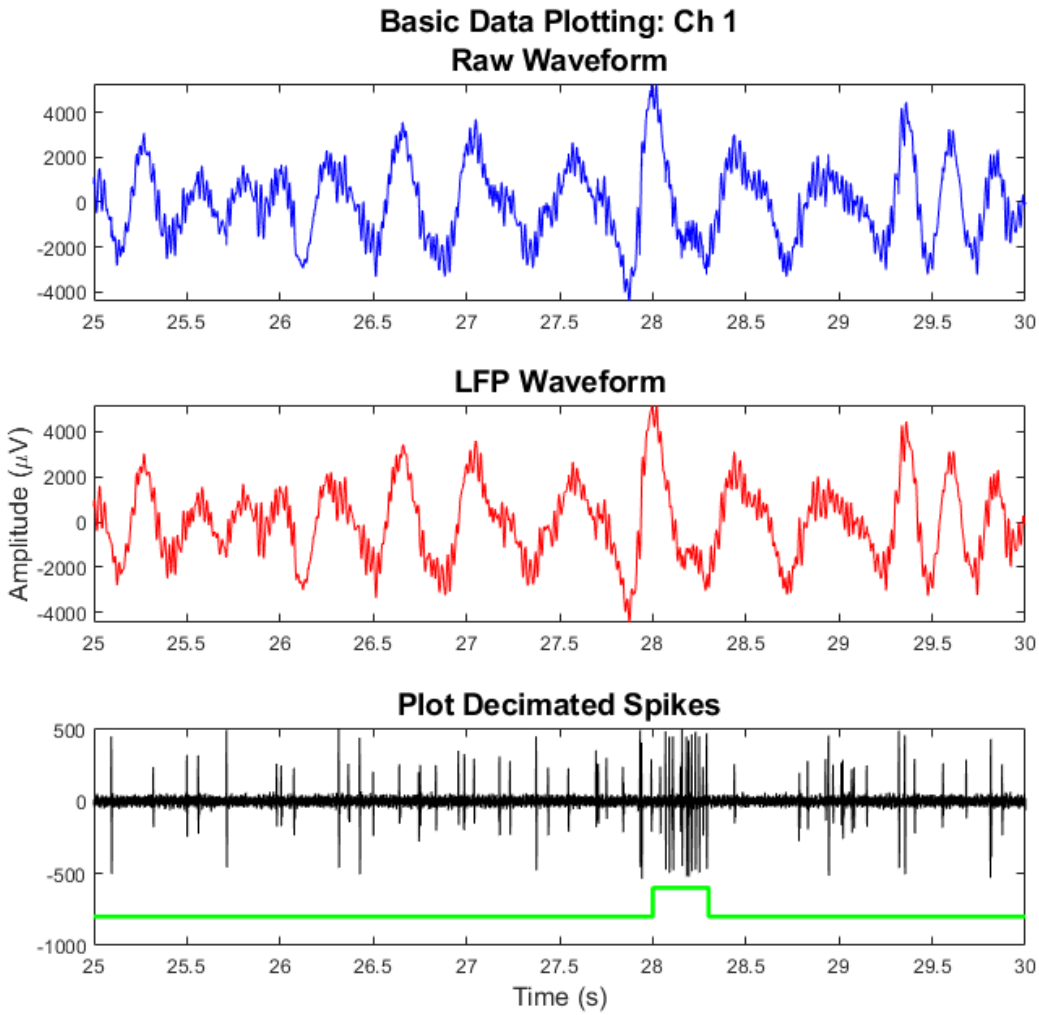
Epoc Events

Generate continuous time series for epoc data using epoc timestamps

```
% StimSync epoc event
STIMSYNC = 'PC0_';
pc0_on = data.epocs.(STIMSYNC).onset;
pc0_off = data.epocs.(STIMSYNC).offset;
pc0_x = reshape(kron([pc0_on, pc0_off], [1, 1])', [], 1);
```

Make a time series waveform of epoc values and plot them.

```
sz = length(pc0_on);
d = data.epocs.(STIMSYNC).data';
pc0_y = reshape([zeros(1, sz); d; d; zeros(1, sz)], 1, []);
hold on; plot(pc0_x, 200*(pc0_y) - 800, 'g', 'LineWidth', 2);
```


Basic Data Plotting: Ch 1
Raw Waveform
4000
2000
0
-2000
-4000
25
25.5
26
26.5
27
27.5
28
28.5
29
29.5
30
LFP Waveform
Amplitude (μV)
4000
2000
0
-2000
-4000
25
25.5
26
26.5
27
27.5
28
28.5
29
29.5
30
Plot Decimated Spikes
500
0
-500
-1000
25
25.5
26
26.5
27
27.5
28
28.5
29
29.5
30
Time (s)

Fiber Photometry Epoch Averaging Example

This example goes through fiber photometry analysis using techniques such as data smoothing, bleach detrending, and z-score analysis.
The epoch averaging was done using TDTfilter.

Author Contributions:
TDT, David Root, and the Morales Lab contributed to the writing and/or conceptualization of the code.
The signal processing pipeline was inspired by the workflow developed by David Barker et al. (2017) for the Morales Lab.
The data used in the example were provided by David Root.

Author Information:
David H. Root
Assistant Professor
Department of Psychology & Neuroscience
University of Colorado, Boulder
Lab Website: https://www.root-lab.org
david.root@colorado.edu

About the authors:
The Root lab and Morales lab investigate the neurobiology of reward, aversion, addiction, and depression.

TDT edits all user submissions in coordination with the contributing author(s) prior to publishing.

Download M File

Housekeeping

Clear workspace and close existing figures. Add SDK directories to MATLAB path.

```
close all; clear all; clc;
[MAINEXAMPLEPATH,name,ext] = fileparts(cd); % \TDTMatlabSDK\Examples
DATAPATH = fullfile(MAINEXAMPLEPATH, 'ExampleData'); %
\TDTMatlabSDK\Examples\ExampleData
[SDKPATH,name,ext] = fileparts(MAINEXAMPLEPATH); % \TDTMatlabSDK
addpath(genpath(SDKPATH));
```

Importing the Data

This example assumes you downloaded our example data sets and extracted it into the \TDTMatlabSDK\Examples directory. To import your own data, replace 'BLOCKPATH' with the path to your own data block.

In Synapse, you can find the block path in the database. Go to Menu → History. Find your block, then Right-Click → Copy path to clipboard.

```
BLOCKPATH = fullfile(DATAPATH,'FiPho-180416');
```

Setup the variables for the data you want to extract

We will extract two different stream stores surrounding the 'PtAB' epoch event. We are interested in a specific event code for the shock onset.

```
REF_EPOC = 'PtAB'; % event store name. This holds behavioral codes that are
% read through Ports A & B on the front of the RZ
SHOCK_CODE = 64959; % shock onset event code we are interested in
STREAM_STORE1 = 'x4054'; % name of the 405 store
STREAM_STORE2 = 'x4654'; % name of the 465 store
TRANGE = [-10 20]; % window size [start time relative to epoc onset, window
duration]
BASELINE_PER = [-10 -6]; % baseline period within our window
ARTIFACT = Inf; % optionally set an artifact rejection level

% Now read the specified data from our block into a MATLAB structure.
data = TDTbin2mat(BLOCKPATH, 'TYPE', {'epocs', 'scalars', 'streams'});
```

```
read from t=0.00s to t=583.86s
```

Use TDTfilter to extract data around our epoc event

Using the 'TIME' parameter extracts data only from the time range around our epoc event. Use the 'VALUES' parameter to specify allowed values of the REF_EPOC to extract. For stream events, the chunks of data are stored in cell arrays structured as data.streams.(STREAM_STORE1).filtered

```
data = TDTfilter(data, REF_EPOC, 'VALUES', SHOCK_CODE, 'TIME', TRANGE);
```

Optionally remove artifacts. If any waveform is above ARTIFACT level, or below -ARTIFACT level, remove it from the data set.

```
art1 = ~cellfun('isempty', cellfun(@(x) x(x>ARTIFACT), data.streams.
(STREAM_STORE1).filtered, 'UniformOutput',false));
art2 = ~cellfun('isempty', cellfun(@(x) x(x<-ARTIFACT), data.streams.
(STREAM_STORE1).filtered, 'UniformOutput',false));
good = ~art1 & ~art2;
data.streams.(STREAM_STORE1).filtered = data.streams.
(STREAM_STORE1).filtered(good);

art1 = ~cellfun('isempty', cellfun(@(x) x(x>ARTIFACT), data.streams.
(STREAM_STORE2).filtered, 'UniformOutput',false));
art2 = ~cellfun('isempty', cellfun(@(x) x(x<-ARTIFACT), data.streams.
(STREAM_STORE2).filtered, 'UniformOutput',false));
good2 = ~art1 & ~art2;
data.streams.(STREAM_STORE2).filtered = data.streams.
(STREAM_STORE2).filtered(good2);

numArtifacts = sum(~good) + sum(~good2);
```

Applying a time filter to a uniformly sampled signal means that the length of each segment could vary by one sample. Let's find the minimum length so we can trim the excess off before calculating the mean.

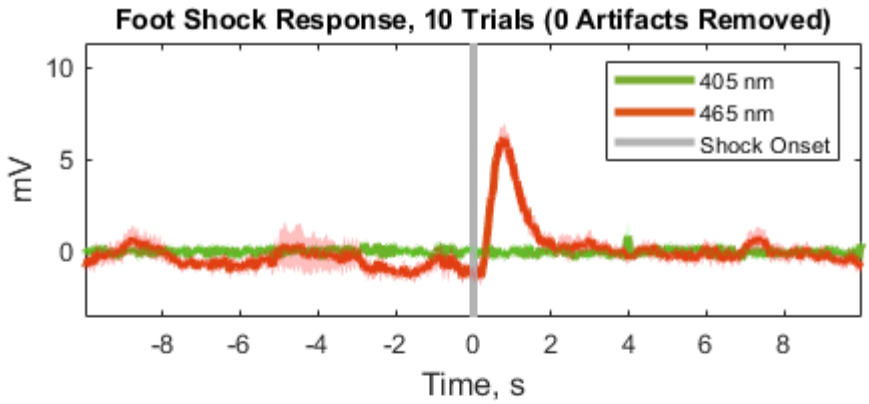
```
minLength1 = min(cellfun('prodofsize', data.streams.
(STREAM_STORE1).filtered));
minLength2 = min(cellfun('prodofsize', data.streams.
(STREAM_STORE2).filtered));
data.streams.(STREAM_STORE1).filtered = cellfun(@(x) x(1:minLength1),
data.streams.(STREAM_STORE1).filtered, 'UniformOutput',false);
data.streams.(STREAM_STORE2).filtered = cellfun(@(x) x(1:minLength2),
data.streams.(STREAM_STORE2).filtered, 'UniformOutput',false);

allSignals = cell2mat(data.streams.(STREAM_STORE1).filtered');

% downsample 10x and average 405 signal
N = 10;
F405 = zeros(size(allSignals(:,1:N:end-N+1)));
for ii = 1:size(allSignals,1)
    F405(ii,:) = arrayfun(@(i) mean(allSignals(ii,i:i+N-1)),
1:N:length(allSignals)-N+1);
end
minLength1 = size(F405,2);

% Create mean signal, standard error of signal, and DC offset of 405 signal
meanSignal1 = mean(F405);
stdSignal1 = std(double(F405))/sqrt(size(F405,1));
dcSignal1 = mean(meanSignal1);

% downsample 10x and average 465 signal
allSignals = cell2mat(data.streams.(STREAM_STORE2).filtered');
F465 = zeros(size(allSignals(:,1:N:end-N+1)));
for ii = 1:size(allSignals,1)
    F465(ii,:) = arrayfun(@(i) mean(allSignals(ii,i:i+N-1)),
1:N:length(allSignals)-N+1);
end
minLength2 = size(F465,2);

% Create mean signal, standard error of signal, and DC offset of 465 signal
meanSignal2 = mean(F465);
stdSignal2 = std(double(F465))/sqrt(size(F465,1));
dcSignal2 = mean(meanSignal2);
```

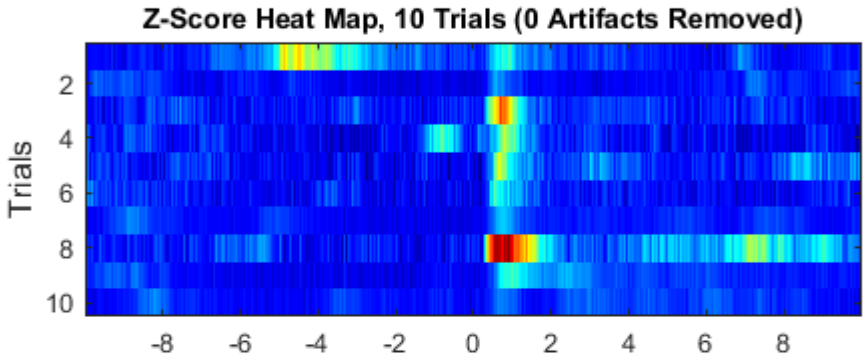
Plot Epoch Averaged Response

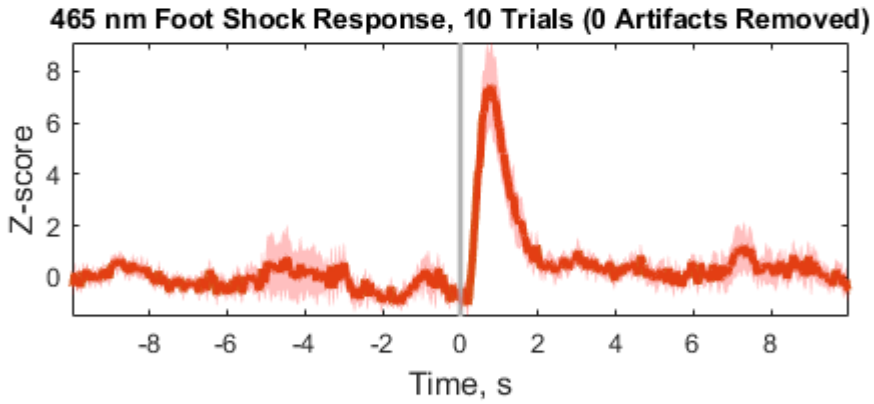
```
% Create the time vector for each stream store
ts1 = TRANGE(1) + (1:minLength1) / data.streams.(STREAM_STORE1).fs*N;
ts2 = TRANGE(1) + (1:minLength2) / data.streams.(STREAM_STORE2).fs*N;

% Subtract DC offset to get signals on top of one another
meanSignal1 = meanSignal1 - dcSignal1;
meanSignal2 = meanSignal2 - dcSignal2;

% Plot the 405 and 465 average signals
figure;
subplot(4,1,1)
plot(ts1, meanSignal1, 'color',[0.4660, 0.6740, 0.1880], 'LineWidth', 3); hold
on;
plot(ts2, meanSignal2, 'color',[0.8500, 0.3250, 0.0980], 'LineWidth', 3);

% Plot vertical line at epoch onset, time = 0
line([0 0], [min(F465(:) - dcSignal2), max(F465(:)) - dcSignal2], 'Color', [.
7 .7 .7], 'LineStyle','-', 'LineWidth', 3)

% Make a legend
legend('405 nm','465 nm','Shock Onset', 'AutoUpdate', 'off');

% Create the standard error bands for the 405 signal
XX = [ts1, fliplr(ts1)];
YY = [meanSignal1 + stdSignal1, fliplr(meanSignal1 - stdSignal1)];

% Plot filled standard error bands.
h = fill(XX, YY, 'g');
set(h, 'facealpha',.25,'edgecolor','none')

% Repeat for 465
XX = [ts2, fliplr(ts2)];
YY = [meanSignal2 + stdSignal2, fliplr(meanSignal2 - stdSignal2)];
h = fill(XX, YY, 'r');
set(h, 'facealpha',.25,'edgecolor','none')

% Finish up the plot
axis tight
xlabel('Time, s','FontSize',12)
ylabel('mV', 'FontSize', 12)
title(sprintf('Foot Shock Response, %d Trials (%d Artifacts Removed)',
numel(data.streams.(STREAM_STORE1).filtered), numArtifacts))
set(gcf, 'Position',[100, 100, 800, 500])

% Heat Map based on z score of 405 fit subtracted 465

% Fitting 405 channel onto 465 channel to detrend signal bleaching
% Scale and fit data
% Algorithm sourced from Tom Davidson's Github:
% https://github.com/tjd2002/tjd-shared-code/blob/master/matlab/photometry/
```

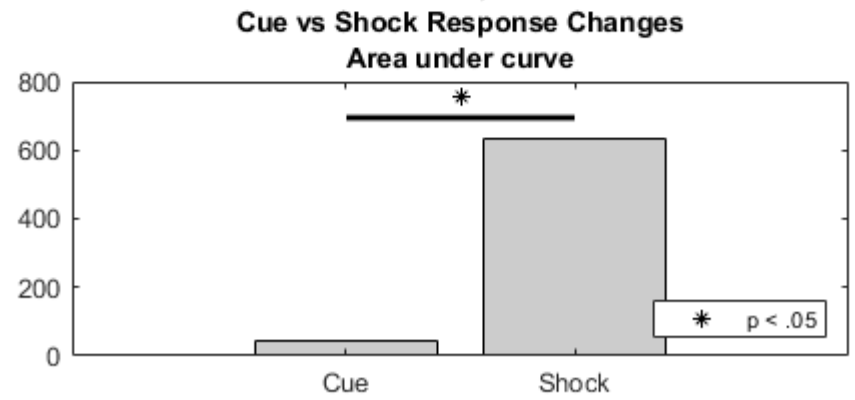
```
FP_normalize.m

bls = polyfit(F405(1:end), F465(1:end), 1);
Y_fit_all = bls(1) .* F405 + bls(2);
Y_dF_all = F465 - Y_fit_all;

zall = zeros(size(Y_dF_all));
for i = 1:size(Y_dF_all,1)
    ind = ts2(1,:) < BASELINE_PER(2) & ts2(1,:) > BASELINE_PER(1);
    zb = mean(Y_dF_all(i,ind)); % baseline period mean (-10sec to -6sec)
    zsd = std(Y_dF_all(i,ind)); % baseline period stdev
    zall(i,:)=(Y_dF_all(i,:) - zb)/zsd; % Z score per bin
end

% Standard error of the z-score
zerror = std(zall)/sqrt(size(zall,1));

% Plot heat map
subplot(4,1,2)

imagesc(ts2, 1, zall);
colormap('jet'); % c1 = colorbar;
title(sprintf('Z-Score Heat Map, %d Trials (%d Artifacts Removed)',
numel(data.streams.(STREAM_STORE1).filtered), numArtifacts));
ylabel('Trials', 'FontSize', 12);

% Fill band values for second subplot. Doing here to scale onset bar
% correctly
XX = [ts2, fliplr(ts2)];
YY = [mean(zall)-zerror, fliplr(mean(zall)+zerror)];

subplot(4,1,3)
plot(ts2, mean(zall), 'color',[0.8500, 0.3250, 0.0980], 'LineWidth', 3); hold
on;
line([0 0], [min(YY), max(YY)], 'Color', [.7 .7 .7], 'LineWidth', 2)

h = fill(XX, YY, 'r');
set(h, 'facealpha',.25,'edgecolor','none')

% Finish up the plot
axis tight
xlabel('Time, s','FontSize',12)
ylabel('Z-score', 'FontSize', 12)
title(sprintf('465 nm Foot Shock Response, %d Trials (%d Artifacts Removed)',
numel(data.streams.(STREAM_STORE1).filtered), numArtifacts))
%c2 = colorbar;

% Quantify changes as area under the curve for cue (-5 sec) and shock (0 sec)
AUC=[]; % cue, shock
AUC(1,1)=trapz(mean(zall(:,ts2(1,:) < -3 & ts2(1,:) > -5)));
AUC(1,2)=trapz(mean(zall(:,ts2(1,:) > 0 & ts2(1,:) < 2)));
```

```
subplot(4,1,4);
hBar = bar(AUC, 'FaceColor', [.8 .8 .8]);

% Run a two-sample T-Test
[h,p,ci,stats] = ttest2(mean(zall(:,ts2(1,:) < -3 & ts2(1,:) >
-5)),mean(zall(:,ts2(1,:) > 0 & ts2(1,:) < 2)));

% Plot significance bar if p < .05
hold on;
centers = get(hBar, 'XData');
plot(centers(1:2), [1 1]*AUC(1,2)*1.1, '-k', 'LineWidth', 2)
p1 = plot(mean(centers(1:2)), AUC(1,2)*1.2, '*k');
set(gca,'xticklabel',{'Cue','Shock'});
title({'Cue vs Shock Response Changes', 'Area under curve'})
legend(p1, 'p < .05','Location','southeast');

set(gcf, 'Position',[100, 100, 500, 1000])
```


Foot Shock Response, 10 Trials (0 Artifacts Removed)
405 nm
465 nm
Shock Onset
mV
Time, s


Z-Score Heat Map, 10 Trials (0 Artifacts Removed)
Trials


465 nm Foot Shock Response, 10 Trials (0 Artifacts Removed)
Z-score
Time, s


Cue vs Shock Response Changes
Area under curve
p < .05
Cue
Shock

Licking Bout Epoc Filtering

This example looks at fiber photometry data in the VTA where subjects are provided sucrose water after a fasting period.
Lick events are captured as TTL pulses.
Objective is to combine many consecutive licking events into a single event based on time difference and lick count thresholds.
New lick bout events can then be used for clear peri-event filtering.

Download M File

Housekeeping

Clear workspace and close existing figures. Add SDK directories to MATLAB path.

```
close all; clear all; clc;

[MAINEXAMPLEPATH,name,ext] = fileparts(cd); % \TDTMatlabSDK\Examples
DATAPATH = fullfile(MAINEXAMPLEPATH, 'ExampleData'); %
\TDTMatlabSDK\Examples\ExampleData
[SDKPATH,name,ext] = fileparts(MAINEXAMPLEPATH); % \TDTMatlabSDK
addpath(genpath(SDKPATH));
```

Importing the Data

This example assumes you downloaded our example data sets and extracted it into the \TDTMatlabSDK\Examples directory. To import your own data, replace 'BLOCKPATH' with the path to your own data block.

In Synapse, you can find the block path in the database. Go to Menu → History. Find your block, then Right-Click → Copy path to clipboard.

```
BLOCKPATH = fullfile(DATAPATH,'VTA4-190125-100559');
```

Call the import function from the MATLAB SDK.

```
data = TDTbin2mat(BLOCKPATH);
```

```
Found Synapse note file: C:
\TDT\TDTMatlabSDK\Examples\ExampleData\VTA4-190125-100559\Notes.txt
read from t=0.00s to t=785.44s
```

Declare data stream and epoc names we will use downstream These are the field names for the relevant streams of the data struct

```
GCAMP = 'x480G';
ISOS = 'x405G';
LICK = 'Ler_';

% Make some pretty colors for later plotting
% <https://math.loyola.edu/~loberbro/matlab/html/colorsInMatlab.html>
red = [0.8500, 0.3250, 0.0980];
green = [0.4660, 0.6740, 0.1880];
cyan = [0.3010, 0.7450, 0.9330];
gray1 = [.7 .7 .7];
gray2 = [.8 .8 .8];
```

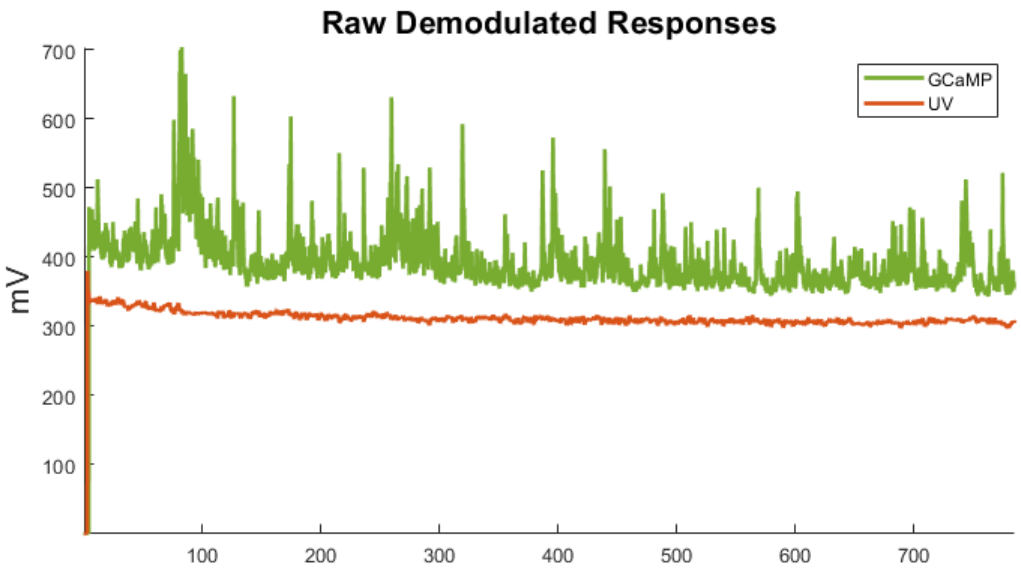
Basic plotting and artifact removal

Make a time array based on number of samples and sample freq of demodulated streams

```
time = (1:length(data.streams.(GCAMP).data))/data.streams.(GCAMP).fs;
```

Plot both unprocessed demodulated data streams

```
figure('Position',[100, 100, 800, 400])
hold on;
p1 = plot(time, data.streams.(GCAMP).data,'color',green,'LineWidth',2);
p2 = plot(time, data.streams.(ISOS).data,'color',red,'LineWidth',2);
title('Raw Demodulated Responses','fontsize',16);
ylabel('mV','fontsize',16);
axis tight;
legend([p1 p2], {'GCaMP','UV'});
```



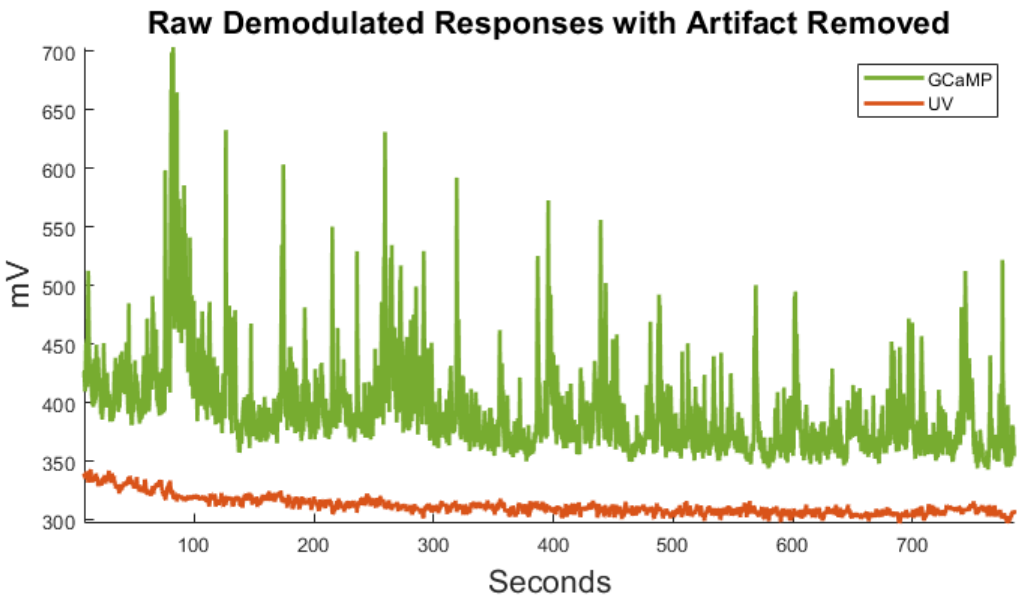
Artifact removal

There is often a large artifact on the onset of LEDs turning on Remove data below a set time t

```
t = 8; % time threshold below which we will discard
ind = find(time>t,1); % find first index of when time crosses threshold
time = time(ind:end); % reformat vector to only include allowed time
data.streams.(GCAMP).data = data.streams.(GCAMP).data(ind:end);
data.streams.(ISOS).data = data.streams.(ISOS).data(ind:end);
```

Plot again at new time range

```
clf;
hold on;
p1 = plot(time, data.streams.(GCAMP).data,'color',green,'LineWidth',2);
p2 = plot(time, data.streams.(ISOS).data,'color',red,'LineWidth',2);
title('Raw Demodulated Responses with Artifact Removed','fontsize',16);
xlabel('Seconds','fontsize',16)
ylabel('mV','fontsize',16);
axis tight;
legend([p1 p2], {'GCaMP','UV'});
```



Downsample data doing local averaging

Average around every Nth point and downsample Nx

```
N = 10; % multiplicative for downsampling
data.streams.(GCAMP).data = arrayfun(@(i)...
    mean(data.streams.(GCAMP).data(i:i+N-1)),...
    1:N:length(data.streams.(GCAMP).data)-N+1);
data.streams.(ISOS).data = arrayfun(@(i)...
    mean(data.streams.(ISOS).data(i:i+N-1)),...
    1:N:length(data.streams.(ISOS).data)-N+1);
```

Decimate time array and match length to demodulated stream

```
time = time(1:N:end);
time = time(1:length(data.streams.(GCAMP).data));
```

Detrending and dFF

```
bls = polyfit(data.streams.(ISOS).data,data.streams.(GCAMP).data,1);
Y_fit_all = bls(1) .* data.streams.(ISOS).data + bls(2);
Y_dF_all = data.streams.(GCAMP).data - Y_fit_all; %dF (units mV) is not dFF
```

Full dFF according to Lerner et al. 2015 https://dx.doi.org/10.1016/j.cell.2015.07.014 dFF using 405 fit as baseline

```
dFF = 100*(Y_dF_all)./Y_fit_all;
std_dFF = std(double(dFF));
```

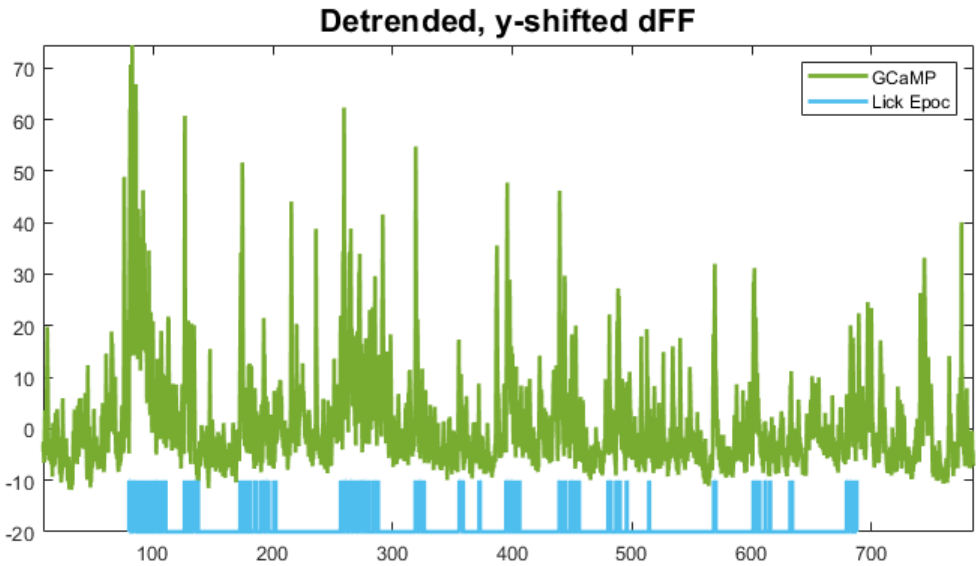
Turn Licking Events into Lick Bouts

Make a continuous time series of Licking TTL events (epocs) and plot

```
LICK_on = data.epocs.(LICK).onset;
LICK_off = data.epocs.(LICK).offset;
LICK_x = reshape(kron([LICK_on, LICK_off], [1, 1])', [], 1);
sz = length(LICK_on);
d = data.epocs.(LICK).data';
y_scale = 10; %adjust according to data needs
y_shift = -20; %scale and shift are just for aesthetics
LICK_y = reshape([zeros(1, sz); d; d; zeros(1, sz)], 1, []);
```

First subplot in a series: dFF with lick epocs

```
figure('Position',[100, 100, 800, 400]);
p1 = plot(time, dFF, 'Color',green,'LineWidth',2); hold on;
p2 = plot(LICK_x, y_scale*(LICK_y) + y_shift,'color',cyan,'LineWidth',2);
title('Detrended, y-shifted dFF','fontsize',16);
legend([p1 p2],'GCaMP','Lick Epoc');
axis tight
```



Now combine lick epocs that happen in close succession to make a single on/off event (a lick BOUT). Top view logic: if difference between consecutive lick onsets is below a certain time

threshold and there was more than one lick in a row, then consider it as one bout, otherwise it is its own bout. Also, make sure a minimum number of licks was reached to call it a bout.

Make a new epoc event in the data structure

```
LICK_EVENT = 'Lick_Event';
data.epocs.(LICK_EVENT).name = LICK_EVENT;
data.epocs.(LICK_EVENT).onset = [];
data.epocs.(LICK_EVENT).offset = [];
data.epocs.(LICK_EVENT).typeStr = data.epocs.(LICK).typeStr;
data.epocs.(LICK_EVENT).data = [];
```

Find differences in onsets and threshold for major difference indices

```
lick_on_diff = diff(data.epocs.(LICK).onset);
BOUT_TIME_THRESHOLD = 10; % example bout time threshold, in seconds
lick_diff_ind = find(lick_on_diff >= BOUT_TIME_THRESHOLD);
```

Make an onset/ offset array based on threshold indices

```
diff_ind_i = 1;
for i = 1:length(lick_diff_ind)
    % BOUT onset is thresholded onset index of lick epoc event
    data.epocs.(LICK_EVENT).onset(i) = data.epocs.(LICK).onset(diff_ind_i);

    % BOUT offset is thresholded offset of lick event before next onset
    data.epocs.(LICK_EVENT).offset(i) = ...
        data.epocs.(LICK).offset(lick_diff_ind(i));

    data.epocs.(LICK_EVENT).data(i) = 1; % set the data value, arbitrary 1
    diff_ind_i = lick_diff_ind(i) + 1; % increment the index
end

% Special case for last event to handle lick event offset indexing
data.epocs.(LICK_EVENT).onset = [data.epocs.(LICK_EVENT).onset, ...
   data.epocs.(LICK).onset(lick_diff_ind(end)+1)];
data.epocs.(LICK_EVENT).offset = [data.epocs.(LICK_EVENT).offset, ...
   data.epocs.(LICK).offset(end)];
data.epocs.(LICK_EVENT).data = [data.epocs.(LICK_EVENT).data, 1];

% Transpose the arrays to make them column vectors like other epocs
data.epocs.(LICK_EVENT).onset = data.epocs.(LICK_EVENT).onset';
data.epocs.(LICK_EVENT).offset = data.epocs.(LICK_EVENT).offset';
data.epocs.(LICK_EVENT).data = data.epocs.(LICK_EVENT).data';

% Note that for speed the previous section could be replaced with these three
lines
% data.epocs.(LICK_EVENT).onset = data.epocs.(LICK).onset([1;
lick_diff_ind+1]);
% data.epocs.(LICK_EVENT).offset = data.epocs.(LICK).offset([lick_diff_ind;
end]);
% data.epocs.(LICK_EVENT).data = ones(1, length(data.epocs.
(LICK_EVENT).onset))';
```

Now determine if it was a 'real' bout or not by thresholding by some user-set number of licks in a row

```
MIN_LICK_THRESH = 4; % four licks or more make a bout
licks_array = zeros(length(data.epocs.(LICK_EVENT).onset),1);
for i = 1:length(data.epocs.(LICK_EVENT).onset)
    % Find number of licks in licks_array between onset ond offset of
    % Our new lick BOUT (LICK_EVENT)
    licks_array(i) = numel(find(data.epocs.(LICK).onset >=...
        data.epocs.(LICK_EVENT).onset(i) & data.epocs.(LICK).onset <=...
        data.epocs.(LICK_EVENT).offset(i)));
end
```

Remove onsets, offsets, and data of thrown out events. MATLAB can use booleans for indexing. Cool!

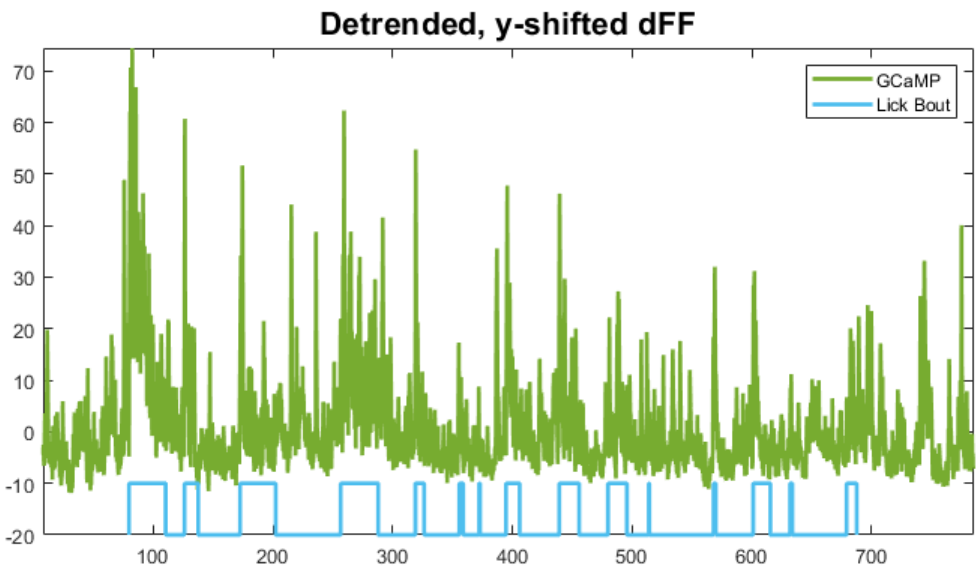
```
data.epocs.(LICK_EVENT).onset((licks_array < MIN_LICK_THRESH)) = [];
data.epocs.(LICK_EVENT).offset((licks_array < MIN_LICK_THRESH)) = [];
data.epocs.(LICK_EVENT).data((licks_array < MIN_LICK_THRESH)) = [];
```

Make continuous time series for lick BOUTS for plotting

```
LICK_EVENT_on = data.epocs.(LICK_EVENT).onset;
LICK_EVENT_off = data.epocs.(LICK_EVENT).offset;
LICK_EVENT_x = reshape(kron([LICK_EVENT_on,LICK_EVENT_off],[1, 1])',[],1);
sz = length(LICK_EVENT_on);
d = data.epocs.(LICK_EVENT).data';
LICK_EVENT_y = reshape([zeros(1, sz); d; d; zeros(1, sz)], 1, []);
```

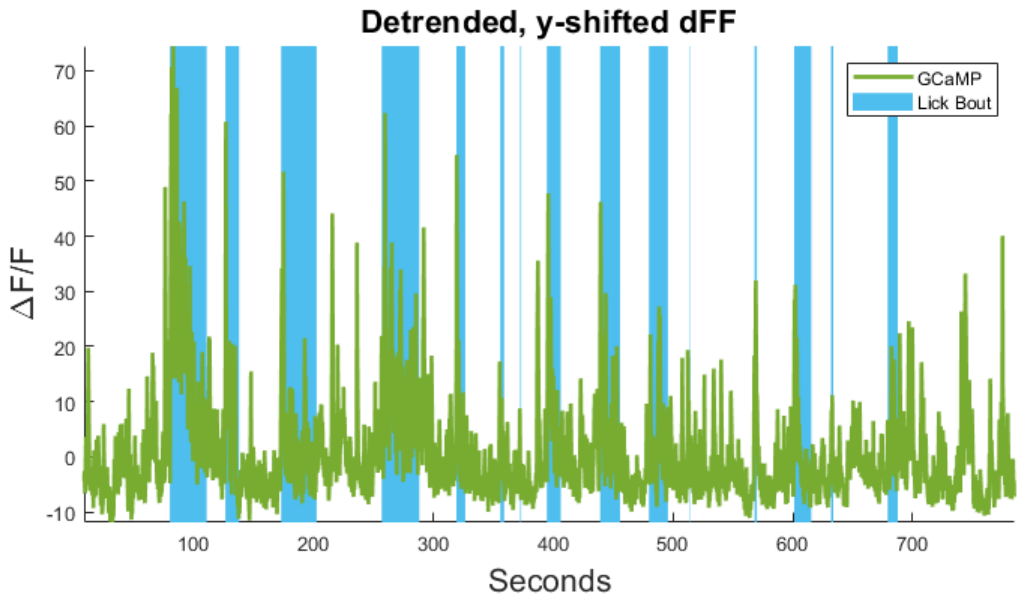
Next step: dFF with newly defined lick bouts

```
clf;
p1 = plot(time, dFF,'Color',green,'LineWidth',2);
hold on;
p2 = plot(LICK_EVENT_x, y_scale*(LICK_EVENT_y) + y_shift,...
    'color',cyan,'LineWidth',2);
title('Detrended, y-shifted dFF','fontsize',16);
legend([p1 p2],'GCaMP', 'Lick Bout');
axis tight
```



Making nice area fills instead of epocs for aesthetics. Newer versions of MATLAB can use alpha on area fills, which could be desirable

```
clf;
hold on;
dFF_min = min(dFF);
dFF_max = max(dFF);
for i = 1:numel(data.epocs.(LICK_EVENT).onset)
    h1(i) = area([data.epocs.(LICK_EVENT).onset(i) ...
        data.epocs.(LICK_EVENT).offset(i)], [dFF_max dFF_max], ...
        dFF_min, 'FaceColor',cyan,'edgecolor', 'none');
end
p1 = plot(time, dFF,'Color',green,'LineWidth',2);
title('Detrended, y-shifted dFF','fontsize',16);
legend([p1 h1(1)],'GCaMP', 'Lick Bout');
ylabel('\DeltaF/F','fontsize',16)
xlabel('Seconds','fontsize',16);
axis tight
```



Time Filter Around Lick Bout Epocs

Note that we are using dFF of the full time-series, not peri-event dFF where f0 is taken from a pre-event baseline period. That is done in another fiber photometry data analysis example.

```
PRE_TIME = 5; % ten seconds before event onset
POST_TIME = 10; % ten seconds after
fs = data.streams.(GCAMP).fs/N; % recall we downsampled by N = 100 earlier
% time span for peri-event filtering, PRE and POST
TRANGE = [-1*PRE_TIME*floor(fs),POST_TIME*floor(fs)];
```

Pre-allocate memory

```
trials = numel(data.epocs.(LICK_EVENT).onset);
dFF_snips = cell(trials,1);
array_ind = zeros(trials,1);
pre_stim = zeros(trials,1);
post_stim = zeros(trials,1);
```

Make stream snips based on trigger onset

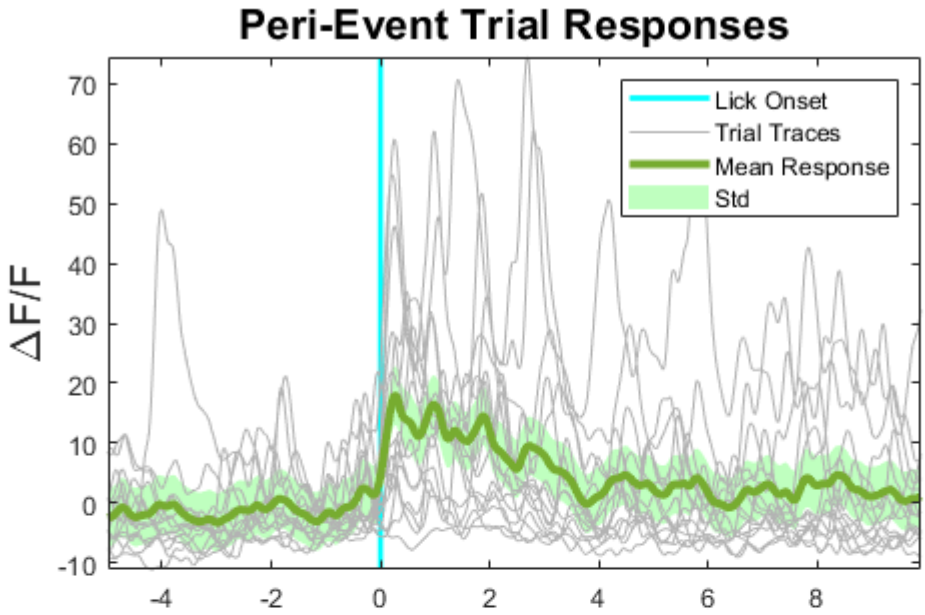
```
for i = 1:trials
    % If the bout cannot include pre-time seconds before event, make zero
    if data.epocs.(LICK_EVENT).onset(i) < PRE_TIME
        dFF_snips{i} = single(zeros(1,(TRANGE(2)-TRANGE(1))));
        continue
    else
        % Find first time index after bout onset
        array_ind(i) = find(time > data.epocs.(LICK_EVENT).onset(i),1);

        % Find index corresponding to pre and post stim durations
        pre_stim(i) = array_ind(i) + TRANGE(1);
        post_stim(i) = array_ind(i) + TRANGE(2);
        dFF_snips{i} = dFF(pre_stim(i):post_stim(i));
    end
end
```

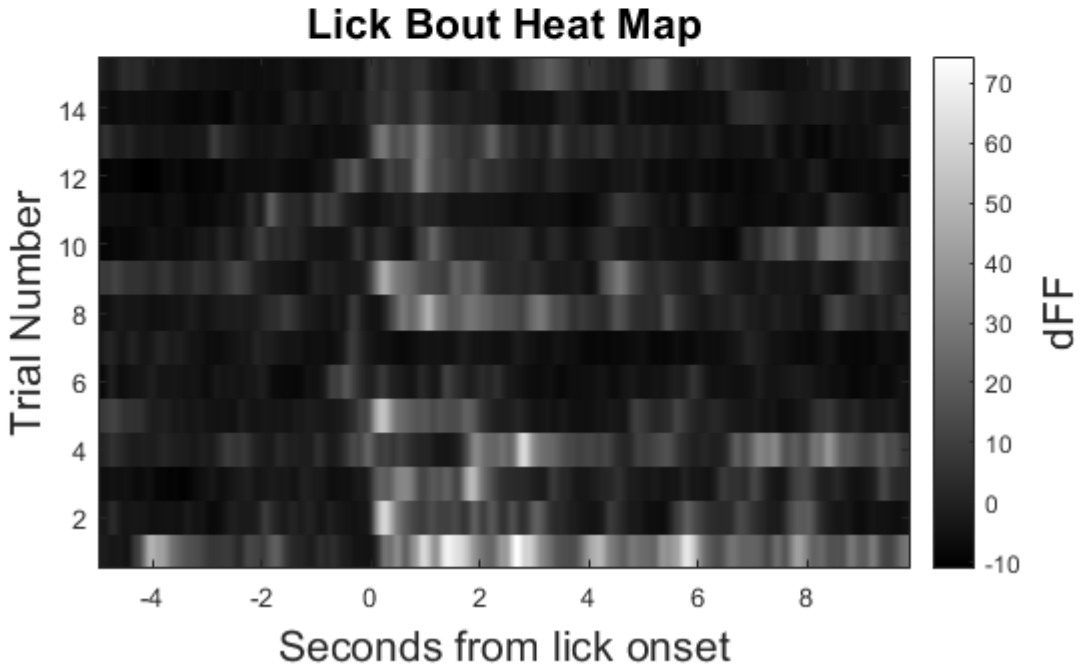
Make all snippet cells the same size based on minimum snippet length

```
minLength = min(cellfun('prodofsize', dFF_snips));
dFF_snips = cellfun(@(x) x(1:minLength), dFF_snips, 'UniformOutput',false);

% Convert to a matrix and get mean
allSignals = cell2mat(dFF_snips);
mean_allSignals = mean(allSignals);
std_allSignals = std(mean_allSignals);

% Make a time vector snippet for peri-events
peri_time = (1:length(mean_allSignals))/fs - PRE_TIME;
```

Make a Peri-Event Stimulus Plot and Heat Map

```
% Make a standard deviation fill for mean signal
figure('Position',[100, 100, 600, 750])
subplot(2,1,1)
xx = [peri_time, fliplr(peri_time)];
yy = [mean_allSignals + std_allSignals,...
    fliplr(mean_allSignals - std_allSignals)];
h = fill(xx, yy, 'g'); % plot this first for overlay purposes
hold on;
set(h, 'facealpha', 0.25, 'edgecolor', 'none');

% Set specs for min and max value of event line.
% Min and max of either std or one of the signal snip traces
linemin = min(min(min(allSignals)),min(yy));
linemax = max(max(max(allSignals)),max(yy));

% Plot the line next
l1 = line([0 0], [linemin, linemax],...
    'color','cyan', 'LineStyle', '-', 'LineWidth', 2);
% Plot the signals and the mean signal
p1 = plot(peri_time, allSignals', 'color', gray1);
p2 = plot(peri_time, mean_allSignals, 'color', green, 'LineWidth', 3);
hold off;

% Make a legend and do other plot things
legend([l1, p1(1), p2, h],...
    {'Lick Onset','Trial Traces','Mean Response','Std'},...
    'Location','northeast');
title('Peri-Event Trial Responses','fontsize',16);
ylabel('\DeltaF/F','fontsize',16);
axis tight;
% Make an invisible colorbar so this plot aligns with one below it
temp_cb = colorbar('Visible', 'off');

% Heat map
subplot(2,1,2)
imagesc(peri_time, 1, allSignals); % this is the heatmap
set(gca,'YDir','normal') % put the trial numbers in better order on y-axis
colormap(gray()) % colormap otherwise defaults to perula
title('Lick Bout Heat Map','fontsize',16)
ylabel('Trial Number','fontsize',16)
xlabel('Seconds from lick onset','fontsize',16)
cb = colorbar;
ylabel(cb, 'dFF','fontsize',16)
axis tight;
```


Peri-Event Trial Responses
Lick Onset
Trial Traces
Mean Response
Std
ΔF/F
70
60
50
40
30
20
10
0
-10
-4
-2
0
2
4
6
8


Lick Bout Heat Map
Trial Number
14
12
10
8
6
4
2
-4
-2
0
2
4
6
8
Seconds from lick onset
dFF
70
60
50
40
30
20
10
0
-10

Export Continuous Data To Binary File

Import continuous data into MATLAB using TDTbin2mat
Export the data to a binary file (f32 floating point or 16-bit integer)
Channels are interlaced in the final output file
Good for exporting to other data analysis applications

Download M File

Housekeeping

Clear workspace and close existing figures. Add SDK directories to MATLAB path.

```
close all; clear all; clc;
[MAINEXAMPLEPATH,name,ext] = fileparts(cd); % \TDTMatlabSDK\Examples
DATAPATH = fullfile(MAINEXAMPLEPATH, 'ExampleData'); %
\TDTMatlabSDK\Examples\ExampleData
[SDKPATH,name,ext] = fileparts(MAINEXAMPLEPATH); % \TDTMatlabSDK
addpath(genpath(SDKPATH));
```

Importing the Data

This example assumes you downloaded our example data sets and extracted it into the \TDTMatlabSDK\Examples directory. To import your own data, replace 'BLOCKPATH' with the path to your own data block.

In Synapse, you can find the block path in the database. Go to Menu → History. Find your block, then Right-Click → Copy path to clipboard.

```
BLOCKPATH = fullfile(DATAPATH,'Algernon-180308-130351');
```

Setup the variables for the data you want to extract

We will extract the stream stores and output them to 16-bit integer files

```
FORMAT = 'i16'; % i16 = 16-bit integer, f32 = 32-bit floating point
SCALE_FACTOR = 1e6; % scale factor for 16-bit integer conversion, so units
are uV
% Note: The recommended scale factor for f32 is 1
```

read the first second of data to get the channel count

```
data = TDTbin2mat(BLOCKPATH, 'TYPE', {'streams'}, 'T2', 1);
```

```
read from t=0.00s to t=1.00s
```

If you want an individual store, use the 'STORE' filter like this:

data = TDTbin2mat(BLOCKPATH, 'TYPE', {'streams'}, 'STORE', 'Wav1', 'T2', 1);

Loop through all the streams and save them to disk in 10 second chunks

```
fff = fields(data.streams);
TIME_DELTA = 10;
for ii = 1:numel(fff)
    T1 = 0;
    T2 = T1 + TIME_DELTA;

    thisStore = fff{ii};
    OUTFILE = fullfile(BLOCKPATH, [thisStore '.' FORMAT]);

    fid = fopen(OUTFILE, 'wb');

    data = TDTbin2mat(BLOCKPATH, 'STORE', fff{ii}, 'T1', T1, 'T2', T2);

    % loop through data in 10 second increments
    while ~isempty(data.streams.(thisStore).data)
        if strcmpi(FORMAT, 'i16')
            fwrite(fid, SCALE_FACTOR*reshape(data.streams.(thisStore).data, 1,
[]), 'integer*2');
        elseif strcmpi(FORMAT, 'f32')
            fwrite(fid, SCALE_FACTOR*reshape(data.streams.(thisStore).data, 1,
[]), 'single');
        else
            warning('Format %s not recognized. Use i16 or f32', FORMAT);
            break
        end
        T1 = T2;
        T2 = T2 + TIME_DELTA;
        data = TDTbin2mat(BLOCKPATH, 'STORE', fff{ii}, 'T1', T1, 'T2', T2);
    end
    fprintf('Wrote %s to output file %s\n', thisStore, OUTFILE);
    fprintf('Sampling Rate: %.6f Hz\n', data.streams.(thisStore).fs);
    fprintf('Num Channels: %d\n', size(data.streams.(thisStore).data, 1));
    fclose(fid);
end
```

```
read from t=0.00s to t=10.00s
read from t=10.00s to t=20.00s
read from t=20.00s to t=30.00s
read from t=30.00s to t=40.00s
read from t=40.00s to t=50.00s
read from t=50.00s to t=60.00s
read from t=60.00s to t=70.00s
read from t=70.00s to t=80.00s
Wrote pNe1 to output file C:
\TDT\TDTMatlabSDK\Examples\ExampleData\Algernon-180308-130351\pNe1.i16
Sampling Rate: 498.246185 Hz
Num Channels: 16
read from t=0.00s to t=10.00s
read from t=10.00s to t=20.00s
read from t=20.00s to t=30.00s
read from t=30.00s to t=40.00s
read from t=40.00s to t=50.00s
read from t=50.00s to t=60.00s
read from t=60.00s to t=70.00s
read from t=70.00s to t=80.00s
Wrote Wav1 to output file C:
\TDT\TDTMatlabSDK\Examples\ExampleData\Algernon-180308-130351\Wav1.i16
Sampling Rate: 24414.062500 Hz
Num Channels: 16
read from t=0.00s to t=10.00s
read from t=10.00s to t=20.00s
read from t=20.00s to t=30.00s
read from t=30.00s to t=40.00s
read from t=40.00s to t=50.00s
read from t=50.00s to t=60.00s
read from t=60.00s to t=70.00s
read from t=70.00s to t=80.00s
Wrote LFP1 to output file C:
\TDT\TDTMatlabSDK\Examples\ExampleData\Algernon-180308-130351\LFP1.i16
Sampling Rate: 3051.757813 Hz
Num Channels: 16
```

Conversion of Continuous Data into Binary Format

This guide is designed to provide TDT data users various options for converting their data into binary format, CSV, EDF, DDT, and other formats for import into third-party applications such as Spike2, Kilosort, Plexon Offline Sorter, or others. Please look through the options to see which one is best suited for your needs.

Scaling Notes

Different applications may have different scaling requirements. Please **always check your scaling** across multiple applications to ensure you are scaling your data properly.

Please compare the TDT block or SEV data in MATLAB or Python or OpenScope (https://www.tdt.com/docs/lightning/openscope/ and https://www.tdt.com/docs/openscope/) vs your end-use application (POS, Spike2, or anything else) to make sure the scaling is correct.

TDT streaming data is almost always stored as 32-bit floating point values in units of Volts. Check the `StoresListing.txt` file inside the block folder for a full listing of the streaming data stores. The information includes: Gizmo name, store name, data format, scale factor, and sampling rate. Here is an example:

```
Object ID : Wave1 - Stream Data Storage
 Store ID : Wav1
  Format  : Float-32
  Scale   : Unity
  Rate    : 24414.1 Hz
```

> **Note**
>
> Note that the sampling rate shown in this file is rounded. The exporting tool will export the exact sampling rate. For TDTexport, this is a text file that is created next to the binary files.

When exporting as `int16`, generally you will use a scale factor of `1e6`. This converts the units to microVolt.

One common example of necessary scaling is when exporting to Plexon Offline Sorter. You may need to adjust the scale in Plexon under the Preamp/Total gain to have a gain of 153, as

shown in the image in the Using OpenBrowser or OpenBridge section. This is because Plexon imports DDT files as 16-bit files with an assumed max resolution of 5 V, which makes the resolution 0.153 uV/bit, but you will probably want to adjust that resolution to be 1 uV / bit. However, import your data without this change first to see if the scaling is off, then adjust accordingly and check again.

Using TDTexport

TDTexport is a command-line tool for exporting continuous data to binary or CSV files. Here is a link to the file you will need: https://www.tdt.com/files/tech/TDTexport.exe (download will begin automatically). Copy this file into `C:\TDT\` and use it from the Windows Command line like this:

```
> C:\TDT\TDTexport.exe C:\TDT\OpenEx\Tanks\DEMOTANK2\Block-2 --scale 1e6 --
dtype i16 --channels 1 3 5 --stores Wav1 Wav2
```

In this example, channels 1, 3, 5 of the Wav1 and Wav2 stores from block `C:\TDT\OpenEx\Tanks\DEMOTANK2\Block-2` are scaled by 1e6 and converted to interlaced i16 files. The files will be named `C:\TDT\OpenEx\Tanks\DEMOTANK2\Block-2\Wav1.16` and `C:\TDT\OpenEx\Tanks\DEMOTANK2\Block-2\Wav2.i16`.

This method will export the data as an interlaced binary file (although you can specify interlaced or individual channels), which applications like Kilosort, Spike2, and Plexon Offline Sorter can directly read. You may need to change the file extension from `.i16` to `.bin` for Kilosort.

It also exports a summary text file that includes the options used for the export and the exact sampling rate of the data stores. Here is an example.

```
Path:   G:\TEMP\Block-81\
ScaleFactor:    1.0
Stores:
    Name:   SU_4
    Freq:   24414.0625
    NChan:  16
```

Here is the full list of options:

```
usage: TDTexport.exe [-h] [--format {csv,binary,interlaced}] [--scale SCALE]
                     [--dtype {i16,f32}] [--stores STORES [STORES ...]]
                     [--channels CHANNELS [CHANNELS ...]] [--outdir OUTDIR]
                     [--verbose {1,0}]
                     blockpath

Export TDT stream data to interlaced binary file, one file per store

positional arguments:
  blockpath             data path to export

optional arguments:
  -h, --help            show this help message and exit
  --format {csv,binary,interlaced}
                        exported data format. (default: interlaced)
                        csv:            data export to comma-separated value files
                                        streams: one file per store, one channel per column
                                        epocs: one column onsets, one column offsets
                        binary: streaming data is exported as raw binary files
                                        one file per channel per store
                        interlaced:     streaming data exported as raw binary files
                                        one file per store, data is interlaced
  --scale SCALE         scale data before export (default: 1.0)
  --dtype {i16,f32}     override data format to export, i16 or f32.
                        otherwise, the data format for each store in tank is used
  --stores STORES [STORES ...]
                        string or list, specify a single store or array of stores to extract
  --channels CHANNELS [CHANNELS ...]
                        integer or list, choose a single channel or array of channels
                        to extract from stream or snippet events
  --outdir OUTDIR       output directory for exported files. Defaults to current block folder
                        if not specified
  --verbose {1,0}       print extra debugging statements (default is 0)
```

Using OpenBrowser or OpenBridge (EDF, DDT, PLX)

Use OpenBrowser for conversion of files into EDF format for import into EDF readers such as Polyman, Sleepsign, and others. You can export single or multiple channels into a single EDF output file.

You can follow along with this Lightning Video and read the Open Browser User Guide.

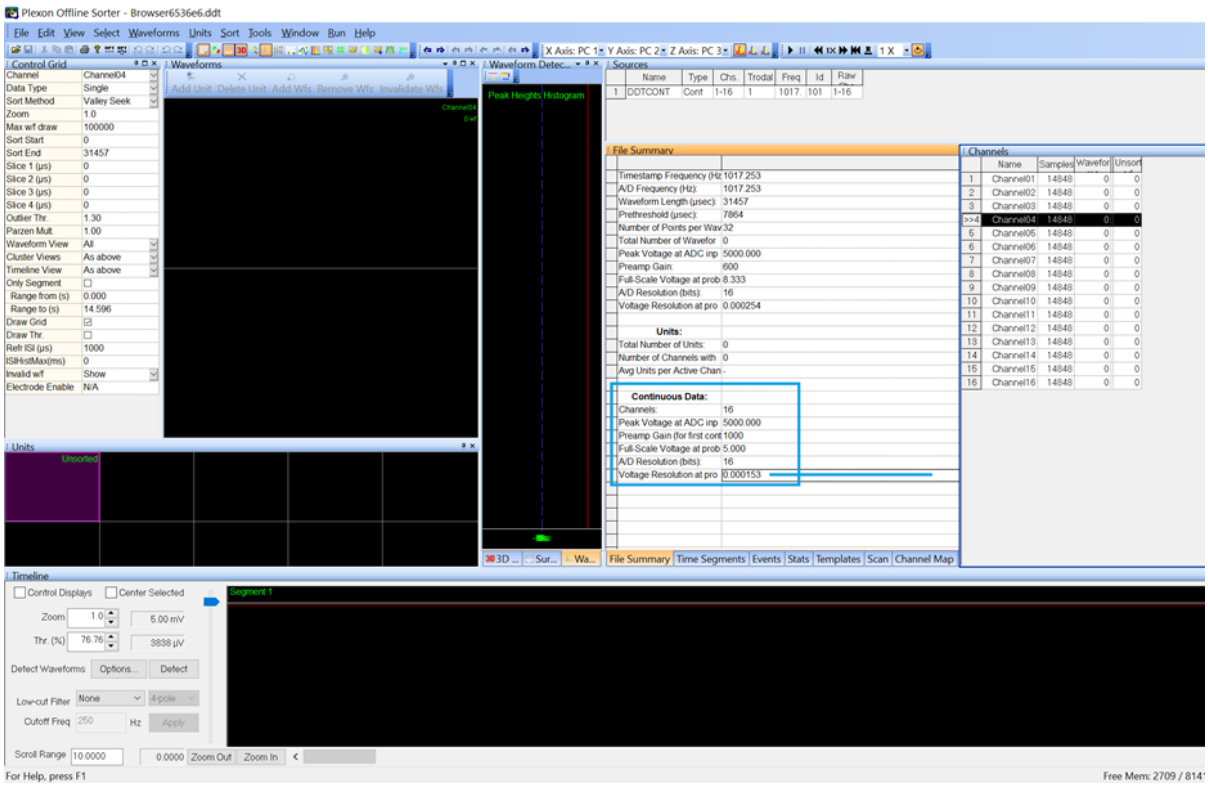
> **Note**
>
> Avoid pressing 'Refresh' in OpenBrowser like the video shows if your data is very long. This could slow down or crash OpenBrowser.

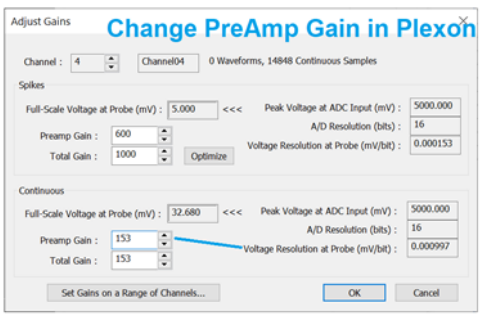
Both OpenBrowser and OpenBridge can convert continuous data into DDT file format, which can be read by Plexon Offline Sorter using the Import and Continuous Binary option (newer versions of POS do not read DDT files directly. It might be simpler to use TDTexport instead, so try both methods if you are unsure.

Here is a link to the OpenBrowser manual exporting section and here is the OpenBridge manual.

DDT files have a data offset of 412 samples, so be sure to use the correct offset when importing.

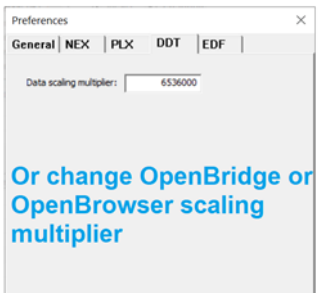
See the Scaling Notes section for important scaling information.





1uV/bit now

One or the other



Using the Python SDK

You can use the python SDK to import block data, SEV files, or subsets of streams or channels using the read_block function, which is a part of the `tdt` module. The `read_block` function also has an export option that you can call. TDTexport is essentially a command line wrapper around this function.

For example:

```
from tdt import read_block

>>> data = read_block('D:\\Tanks\\Scale-210527-121349\
\Subject1-210527-121350', export='interlaced', outdir='D:\\Tanks\
\Scale-210527-121349\\Subject1-210527-121350', prefix='test')
read from t=0s to t=16.71s
exporting TEV stream Wav1...
...100%, 0 seconds elapsed
exporting TEV stream Wav2...
...100%, 0 seconds elapsed
```

This is going to export all the data in streams, so if you want to only export a certain subset be sure to only import that subset to begin with. Use `print(read_block.__doc__)` to see the other options that allow you to select specific stores, time filters, etc, and other export options, like 'binary' and 'csv'.

Using the MATLAB SDK

This binary export script example can be used to export as i16 of f32. Be sure to pick the appropriate scaling for whichever format you select.

For Kilosort, use a scaling factor of 1e6 and export to i16. Then, change the file names from `.i16` to `.bin` and they should work in Kilosort. If you have SEV files, TDTbin2mat will call SEV2mat, so it should work on just the relevant streams if you give it the right Stream Store name. If you use SEV2mat directly, then you may need to move the structure that gets imported into a `data.streams.(StoreName)` structure first.

Video Review and Scoring

This document is meant to provide TDT users with steps for reviewing their data-synchronized videos using OpenScope, third-party media players, or replaying it using a demo MATLAB Script. There is also information for scoring the videos using OpenScope.

The most common synchronized video file that TDT users will have is using a USB Camera but this applies to RV2 video tracker as well.

Notes for Third-Party Video Recording

If you are recording video with a third party software (a common example is EthoVision/ Noldus camera or a camera with GPIO to talk to TDT hardware), then you will want to make sure that you have a way to synchronize your video with your TDT data.

You will not be able to review the video side-by-side your data using OpenScope in this case, but you will still be able to align your video with TDT data if you have a TTL event that timestamps when frames were captured or requested. This most often will be either your Third Party camera sending a TTL pulse to your TDT processor when a frame has been captures (you will save the TTL input in Synapse), or you will send a TTL pulse from Synapse to request a frame to be captured from your external camera.

Best practice would be to start your TDT recording first, then your video, if you are sending TTL pulses externally to TDT, or start your video software first then run TDT if you are sending frame capture requests from TDT to your external camera. These timestamps will then allow you to align the camera frames to your TDT data in post-processing.

View Video Synchronized with Other Data

You can review data alongside video files using OpenScope. Here are several Lightning Videos that demonstrate opening OpenScope:

From within Synapse

Opening OpenScope directly

Observing videos associated with your recordings

Please note that if you are using OpenScope on a machine other than your Synapse computer, or you have relocated your data, then you may need to add in .ini files for Scope to recognize your tanks and block (only needed if Scope is not seeing your data).

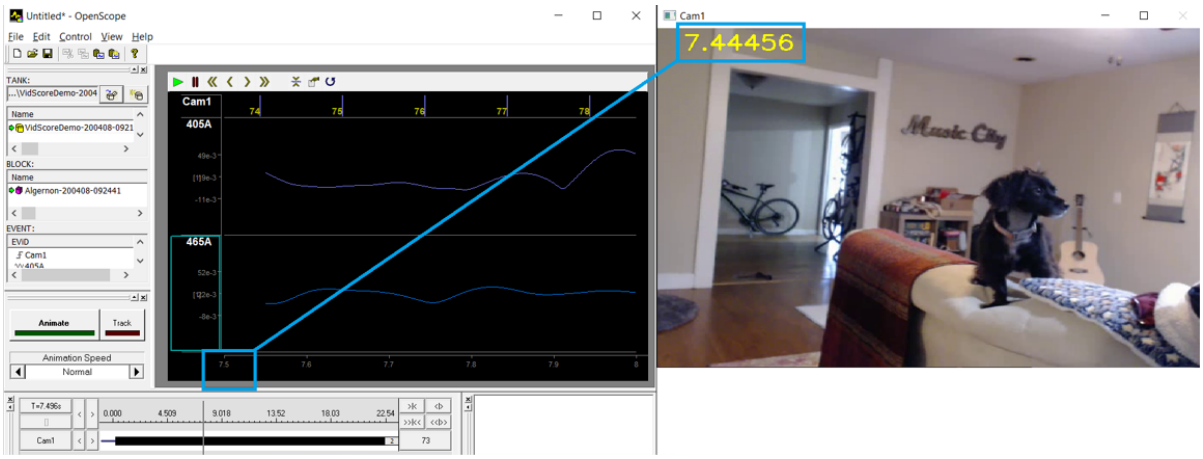
How To Score Videos or Timestamp Data Based on Behavior In Video

To add epoc scoring to your videos: See Video Review and Epoch Annotation in the OpenScope manual.

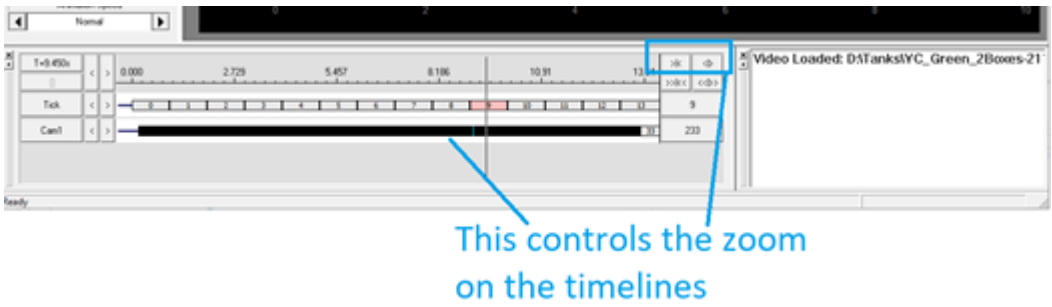
In brief, first you must open the data in OpenScope and then open the associated video. Then open the 'Notes Window'. Once you begin animating the data, if you put the Notes Window into 'Record Mode' then it will begin to track epoc events in the Cam1 epocs (this is the red arrow). With this set up, you can add your notes by manually setting them or using the rapid annotation with the keypad.

Once the notes have been attached to an epoc event (this is the little number next to the epoc mark), they will be put into the data block itself in association with the Cam1 epoc event. Thus, when you import the data into MATLAB using TDTbin2mat and look at the `data.epocs.Cam1` event, you will see that there is a Cam1.notes field as well. These are your scored events.

Below are notes and images that show you how to use the Epoc Annotation/ Notes feature in Open Scope. This allows you to mark your data ("score" your video) to add timestamps to your block where events of interest occur. This can be very helpful for post processing and analysis.

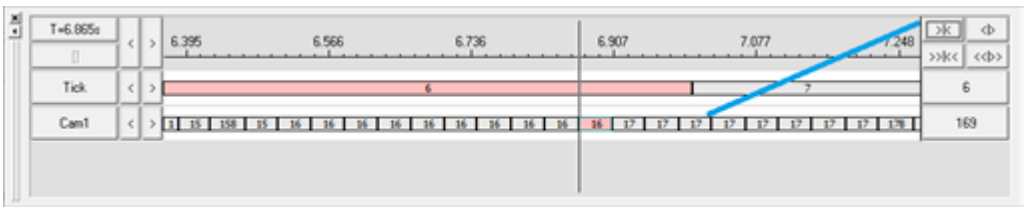


OpenScope Video Frames - video frame timestamp matches epoc 73

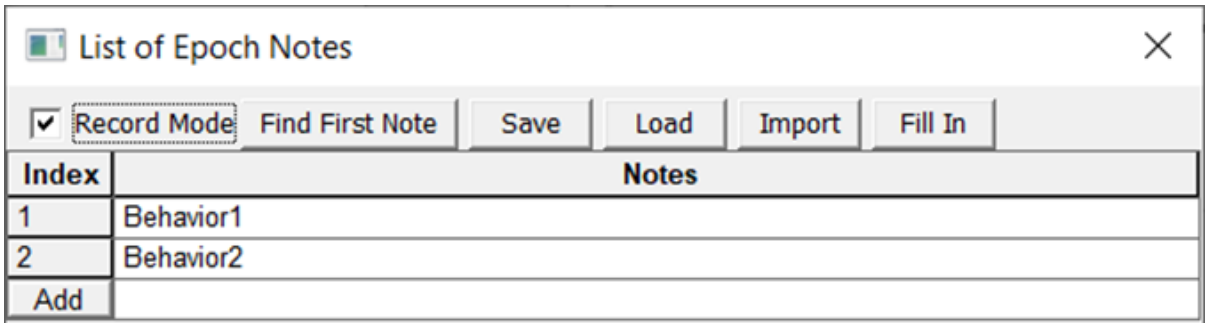


OpenScope Single Frames - zoom on the timelines

If you zoom in far enough you will see individual epoc events, which you can click and move frame by frame. Arrow keys still jump by one span of the x-axis (minimum 0.5 seconds).

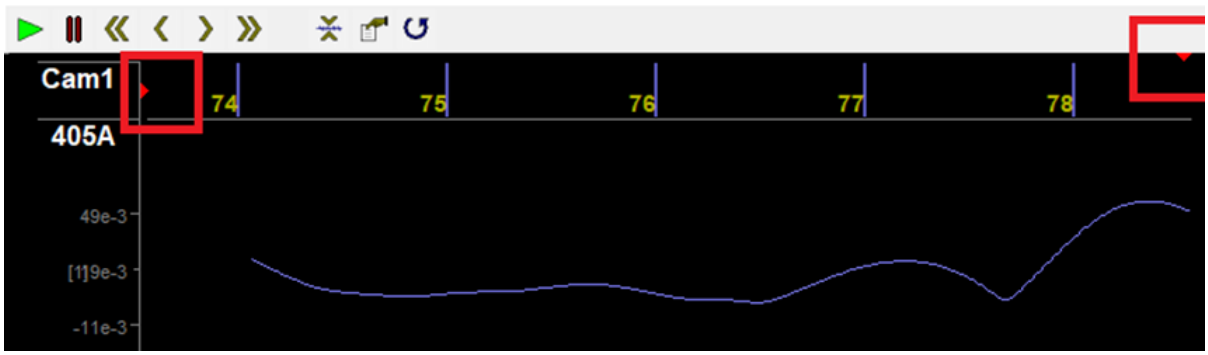


Select View > Notes Window to open the annotation window. From there, add the list of possible notes. Click 'Record Mode' to enable keyboard scoring. Whenever a number key is pressed, the note associated with that number will be attached to the nearest Cam1 epoc event.



OpenScope Notes Window

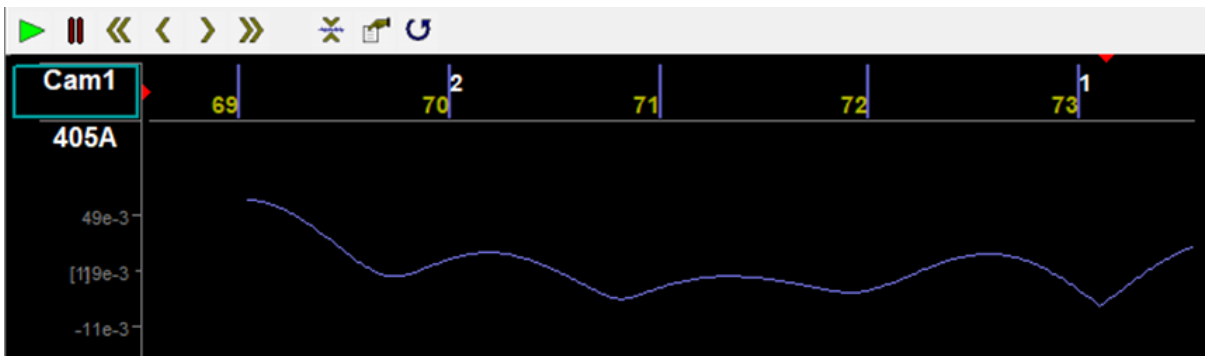
A red arrow marks the currently tracked Cam1 epoc. If you have multiple epoc events, you can change which event gets the notes by right-clicking on the name of it.



OpenScope Notes Attached to Epoc Event

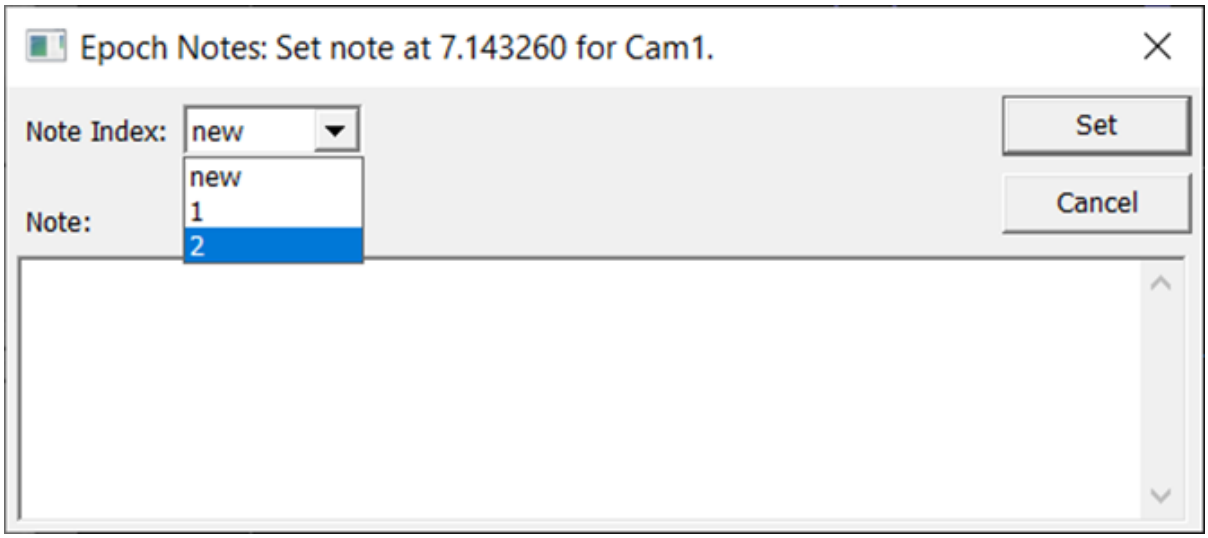
Use arrow keys (turn off "Animate") to move left and right through stream. Note that this will jump by the time span of the x-axis (minimum 0.5 seconds). For finer resolution, you will need to expand the time viewer window and click on individual camera timestamps.

Use Hot Keyboard numbers '1', '2', etc per the Note Index to add a Note to currently tracked epoc (red flag, epoc 73, Note Index 1 in this case).



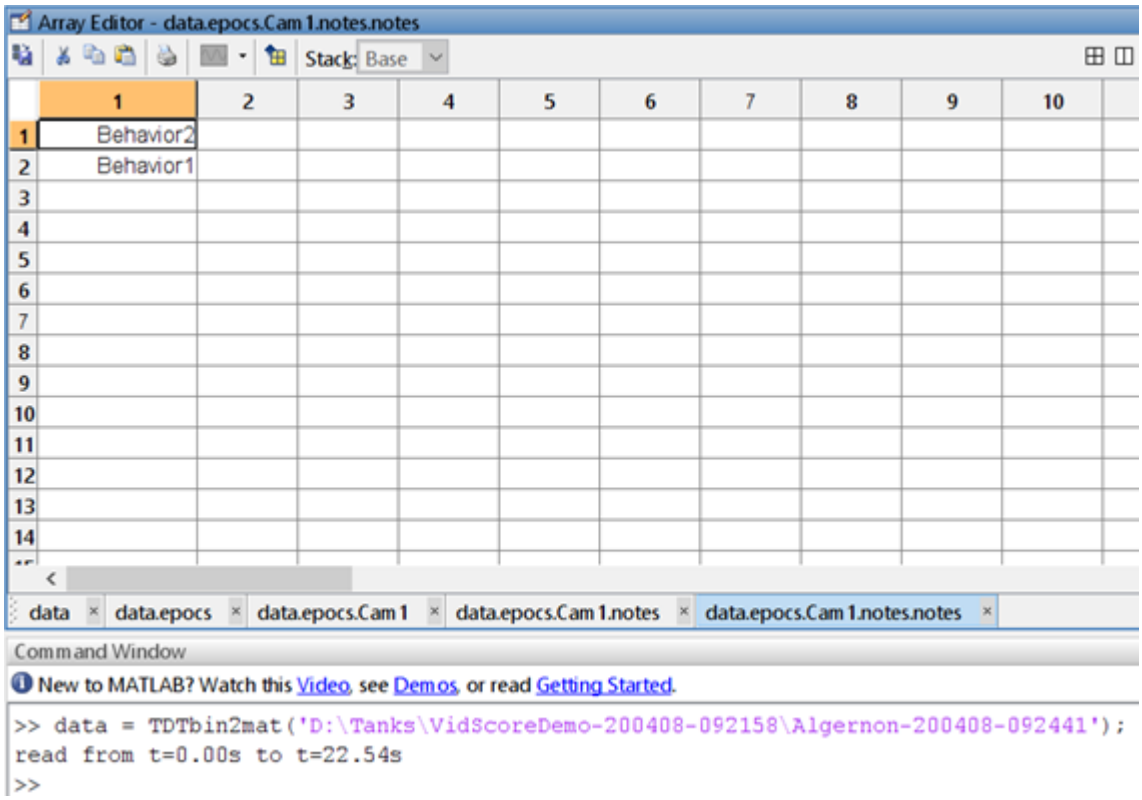
OpenScope Setting Notes

Or right click near Cam1 epoc you want to mark and pick 'Note', then set note accordingly, Epoc 70, Note 2 in this case.



Manually Setting Note

Individual or all notes can be removed. The remaining notes get written to the data block, and can be used for analysis, such as generating peri-event plots.



Note Data Imported Into MATLAB

View Data in Another Video Viewer

VLC Media Player is a wonderful open source media player. There are others, too, that users might want to use to review the video files. The main issue with all of these is that the video players might assume a wrong and static frame rate for the video and the recorded AVI files will not have the same length as the recordings. The most common outcome of this is that the video ends up being ¾ the length of the recorded data upon view.

OpenScope uses the actual Cam1 timestamps to mark when the frame was pulled from the camera. Often, the real camera rate will be 15 fps but a video player will assume 20 fps when reading the data, which is how you get a '45 min' video from a real 1-hour video. If you wanted to read the video in MATLAB or Python to plot alongside your data, then you can use the Cam1 epocs to know the exact time and frame number that corresponds to your data time.

VLC media player can, however, read .sub files that correlate to the real recording time and overlay it on the video. If you want VLC media player to display the timestamp of the video while it is playing, then you can open the AVI file in OpenScope using the video viewer first. This will create a .sub file (in the data block folder) which you can then use in VLC media player to display the timestamps of each frame as it is playing.

Below is MATLAB code that will also generate a sub file. Replace 'block' with your block name, and it will make a file called 'output.sub' in that folder:

```
block = 'F:\TESTAVI';
outfile = [block filesep 'output.sub'];
data = TDTbin2mat(block, 'STORE', 'Cam1');
s = cell(1, length(data.epocs.Cam1.onset));

for i = 1:length(data.epocs.Cam1.onset)
    t = data.epocs.Cam1.onset(i);
    d = data.epocs.Cam1.data(i);
    s{i} = sprintf('{%d}{%d}%.6f\n', d-1, d, t);
end

fid = fopen(outfile, 'w')
fwrite(fid, cell2mat(s));
fclose(fid)
```

Plot My Data Alongside Video

Your TDT data will contain timestamps for video frames as well as the continuous data. You can import data into MATLAB or Python and playback, review, and score your video in that environment.

> **Note**
>
> When importing TDT data and video files into MATLAB or Python, you may notice that the number of camera epoc events does not match the number of video frames.
>
> What you are seeing is normal and is explained in this Tech Note. You can just use the number of frames that matches the length of your Cam1.onset array, starting from index 0 and just chop off the extra video frames at the end.

TDT has an example for overlaying time-locked video with a photometry trace using MATLAB. The script and example data can be found in the MATLAB SDK.

OpenBridge User Guide

Overview

OpenBridge serves as both a general export utility and a "bridge" between an TTank data tank and Plexon's Offline Sorter. OpenBridge exports to NEX v100, DDT v103, EDF+ EDF and PLX v103 file formats. OpenBridge runs on Windows® 10 operating system.

The bridge functionality is available for Plexon's Offline Sorter. It allows for a smooth export and return of sorted data back into the data tank.

Although EDF+ supports exporting snippets, the OpenBridge EDF+ export function does not support exporting snippets. It does support exporting stream data in the EDF+ format.

OpenBridge features:

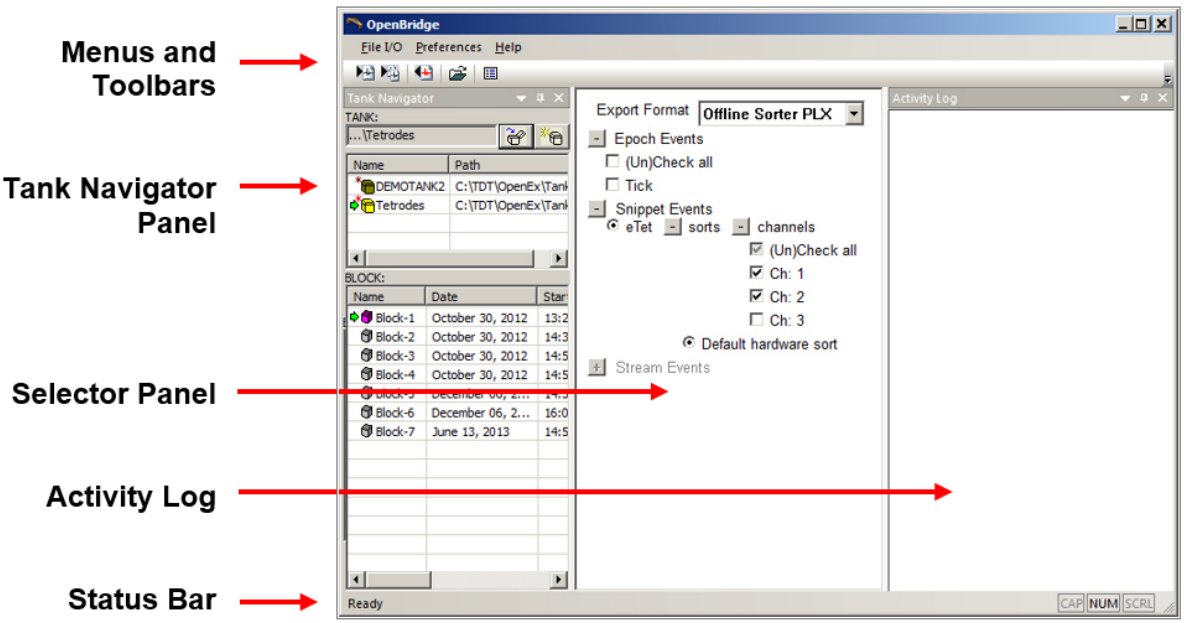
- Allows user selection of events, sorts, and channels for export
- Manages high channel count data sets as multiple files to accommodate Offline Sorter file limitations.
- Automatically launches Offline Sorter on export of PLX or DDT formats.
- Preserves channel numbering across applications.
- Automatically detects changes to PLX files.
- Automatically imports new sorts into the tank.
- Cleans up files after all sorts have been merged into the tank.

See Using the Bridge with Offline Sorter for more information on PLX export and importing sort codes back into the data tank.

Workspace Basics

About the OpenBridge Workspace

OpenBridge provides a versatile workspace where users manage the export process The customizable workspace includes collapsible panels that can be docked or floated and provides auto hide push pin tools. Context sensitive menus, when available, can be accessed by right-clicking the related area.



Menus and Toolbars

Menus and toolbars provide access to all commands and tools. Frequently-used commands are available via toolbar buttons. Hover over a toolbar button to display a description of its function in the Status Bar. [TODO]See The Toolbar and Menus, page 16, for a complete list of commands and tools.

Tank Navigator Panel

A collapsible panel for selecting the tank and block. Registered data tanks are displayed for quick selection.

Selector Panel

This panel is used for selecting export format and data. It shows all available epoch, snippet and stream events for the selected block with check boxes for quick selection of events and channels. Only event types that are allowed for the current export format are available.

Activity Log

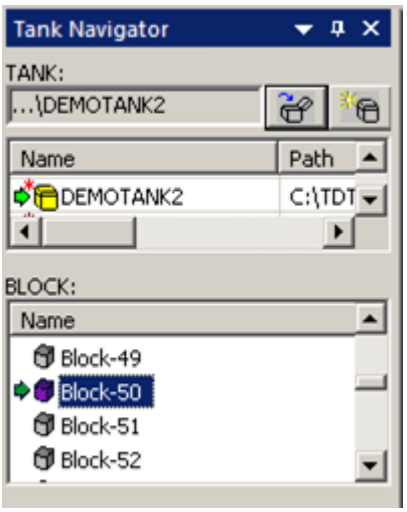
Activity records are displayed, including export/import status and names of exported files.

> **Note**
>
> If a panel is not in view, right-click the menu bar area then select the desired panel on the shortcut menu.

Selecting a Data Set in the Tank Navigator Panel

The tank structure is hierarchical; each tank might include multiple blocks, each block might contain several events and each event might include several channels. OpenBridge exports data as channels. Before the channels can be selected for export, the tank, block, and event(s) must be selected.

The Tank Navigator panel displays tanks and blocks to allow the user to select the desired data set. It can be docked or floated depending on the user's needs.



Selecting a Tank

Users can select a tank in the TANK list area of the Tank Navigator. By default, only registered tanks are displayed. More options, including registering or unregistering tanks, are available from the TANK list shortcut menu.

 Click the Open Existing Tank icon to open the Browse for Tank dialog and navigate to the desired tank.

To register or unregister a tank, right-click the desired tank in the TANK list and select the appropriate option from the shortcut menu.

> **Note**
>
> Unregistering a tank does not delete the tank files, however, only registered tanks will be listed in the TANK list at startup.

Selecting a Block

All blocks in the currently selected tank are shown in the BLOCK list area of the Tank Navigator. Choose one of the blocks for export. This will update the available events in the Selector Panel. The shortcut menu in the BLOCK list controls what information is displayed, making it easier to locate the desired block when the selected Tank contains many blocks.

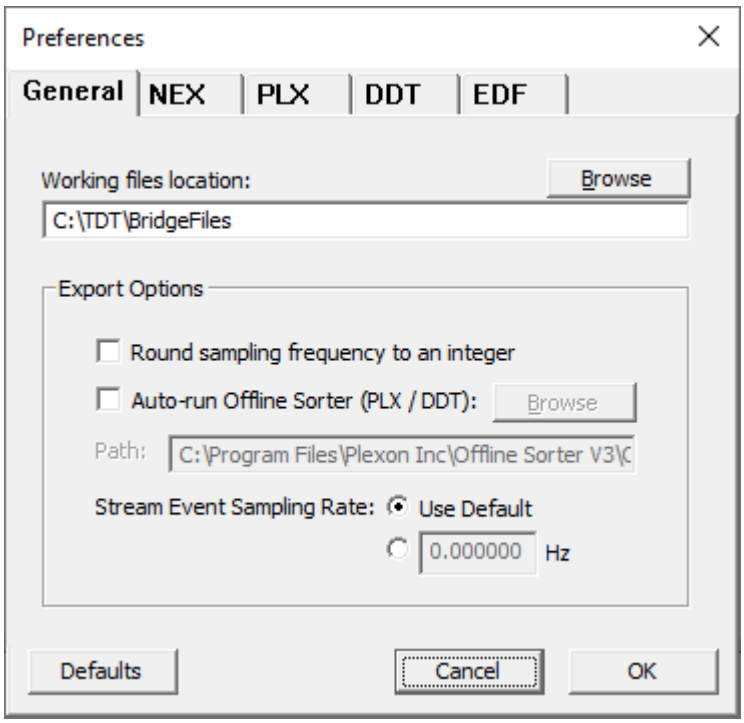
To display the shortcut menu, right-click the BLOCK list:

Option	Description
Refresh	Refresh the BLOCK list
Details	Toggle details view on and off. This shows the block recording date, start time, stop time and duration of each block
Sort	Sort the block list by name instead of start time
Information	Display the Block Notes window, which contains information on all events contained within the selected block

When a block is selected, available Epoch Events, Snippet Events and Stream Events (depending on what export format is selected) are enabled in the Selector Panel.

Setting the Export Preferences

Click *Preferences* in the menu bar. The settings in the Preferences dialog are arranged on tabs, a General tab for options common to more than one export format and individual tabs specific to each export format.



General Tab

Option	Description
Working files location	Set the location where files will be stored during Bridge export/import operations
Round sampling frequency to an integer	Some formats use integer sampling rates by default (e.g. PLX) and others allow floating point sampling rates (e.g. DDT). Check this box to force all export files to use an integer sampling rate. This is useful if you need to align two different data formats later on in your data processing.
Auto-run Offline Sorter (PLX/DDT)	Select to launch exported PLX or DDT file in Offline Sorter immediately after export has completed
Path	Set the path to the Offline Sorter application. This path will vary depending on the operating system and the version of Offline Sorter.
Stream Event Sampling Rate	Select to use the default or specify a sampling rate in the text field

NEX, DDT, EDF Tabs

Option	Description
Data Scaling Multiplier	This scalar is applied during export. A typical value of 1000000 is used when converting floating point uV signals into integer formats.

PLX Tab

Export and import options for the PLX data format. See Using the Bridge with Offline Sorter for more information.

Selecting the Export Format

OpenBridge exports to NEX v100, PLX v103, DDT v103 and EDF+ EDF file formats. The scaling multiplier and other settings for each format are set in the Preferences dialog (see above). Use the Export Format drop-down list in the Selector Panel to choose the desired file type.

> **Note**
>
> See Selecting Events below for information on the data types supported by each export format.

Selecting Events

The selections in the Selector Panel are used to specify data for export. Only event types in the currently selected block that are supported by the selected export format are available. Plus signs to the left of each event type are used to expand events lists. When the selections are not available they appear light gray. The supported event types for each format are as follows:

File Type	Supported Event Types
NEX	Epoch, Snippet, Stream
PLX	Epoch, Snippet
DDT	Stream
EDF	Stream

Use the event tree in the Selector Panel to choose the stores and channels to export.

Exporting Data in NEX, DDT, and EDF Formats

Before exporting, set export format and preferences and select data as described above. Ensure that the desired events, sorts and channel numbers are chosen in the Selector Panel.

Click the *Export selected channels* button on the toolbar, or click "Export Channels" in the File I/O menu.

The exported file name format is:

`TankName_BlockName_SnippetEventName_StreamEventName_Number.fmt`

where `fmt` is the format type and `Number` increments on each subsequent export.

Exporting Data using Batch Processing

Batch processing eliminates the need to configure or apply the export to each block, however, it takes additional processing time. You will need to allow for this extra time in your workflow.

Batching works by applying an export configuration file (*.obconf) to the blocks in a tank. To create the configuration file:

1. Ensure that the desired export format and events, sorts and channel numbers are chosen in the Selector Panel
2. Go the Menu > Configurations and click 'Export'
3. Specify location and file name, then save.

To run a batch export over many blocks in the same tank, modify a command line string as follows:

```
C:\TDT\OpenEx\Bridge\OpenBridge.exe -tank C:\TDT\OpenEx\Tanks\DEMOTANK2 -
configuration C:\TDT\test.obconf -blocks -logPath C:\TDT\BridgeFiles
```

Where:

`C:\TDT\OpenEx\Bridge\OpenBridge.exe` is the path to the OpenBridge application

`C:\TDT\OpenEx\Tanks\DEMOTANK2` is the tank path

`C:\TDT\test.obconf` is the configuration file path

`C:\TDT\BridgeFiles` is the destination path

OpenBridge will use the configuration from the .obconf file to export all blocks in the tank. If you want to export a subset of the blocks in the tank, add the "-blocks" switch with a comma separated list of blocks. For example, add `-blocks "Block-1,Block-2,Block-4"` to the command to export these three blocks only.

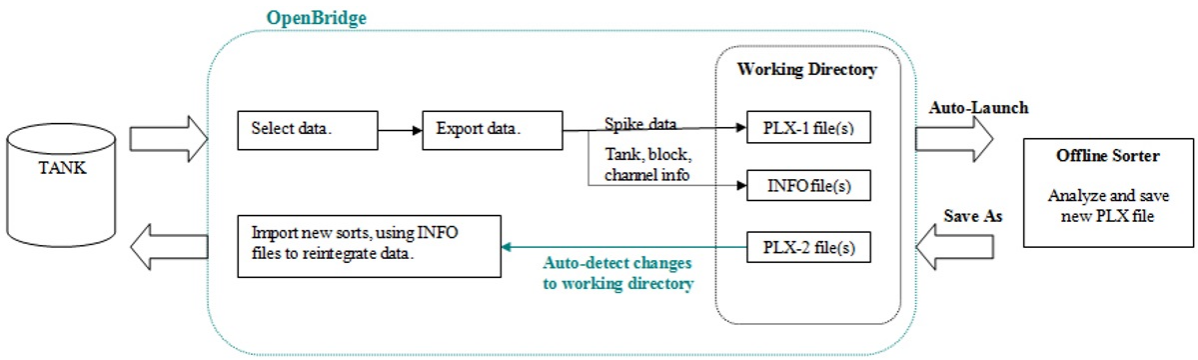
Type or paste the command string into the command line or into the search box on the Start menu to launch. The OpenBridge window will open, perform the export, and then close.

Using the Bridge with Offline Sorter

The OpenBridge exports data to PLX format and can automate and simplify much of the export/import process. It can launch Offline Sorter, manage channel numbering and file handling, and then import data back into the TDT data tank. Work through the steps in this section to make full use of the automated PLX format features in OpenBridge.

When exporting to PLX file format, snippet and epoch data is exported from the tank and analyzed in Offline Sorter to create new sort codes. After the new PLX file is saved in Offline Sorter, OpenBridge can automatically detect when new sort code information is available and reintegrate this new sort code information back into the tank using a new sort name or Sort ID. Once the process is complete, data can be viewed or used for further analysis in OpenSorter, MATLAB or OpenExplorer.

The diagram below illustrates the basic OpenBridge workflow. User preferences control and configure all automated processes.



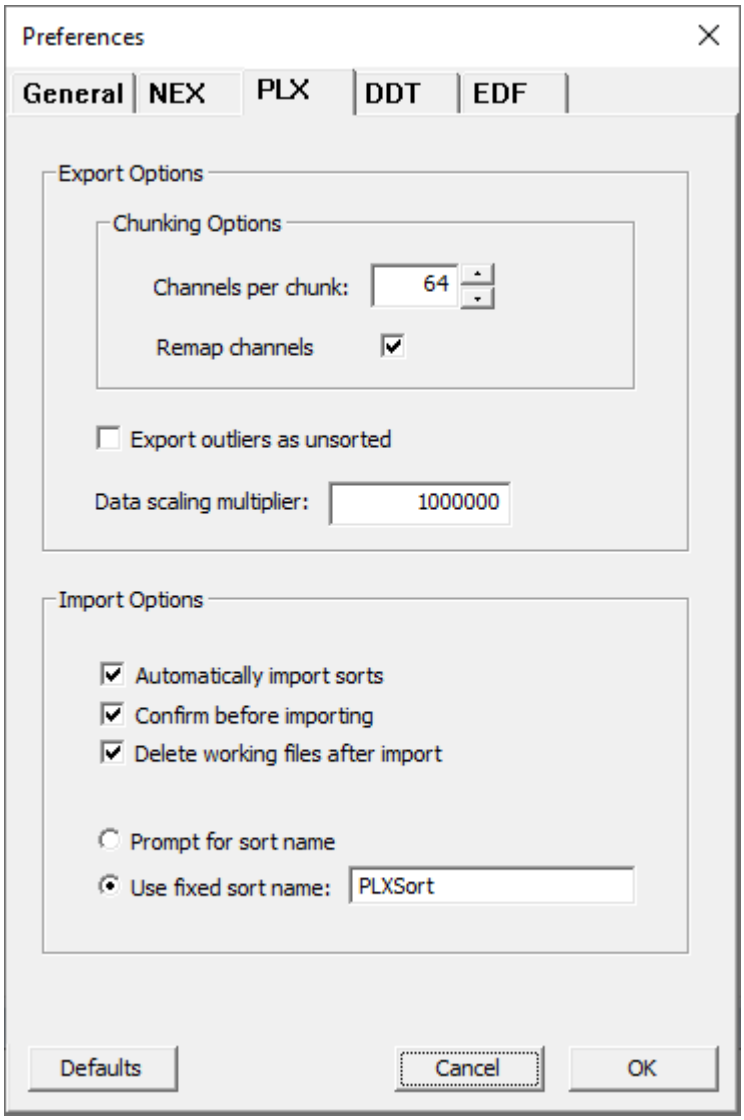
The OpenBridge PLX Workflow

Setting PLX Export Preferences

The PLX export settings are accessed using the PLX tab of the Preferences dialog.

To open the PLX Preferences dialog:

1. Click Preferences on the menu bar.
2. Click the PLX tab.



Export Options	Description
Channels per chunk	Set the number of channels to include in each PLX file during export of high channel count data sets. If exported channel count is higher than this number, multiple files will be exported, each with up to this many channels in it.
Remap channels	Select to renumber channels in sequential order when exporting a subset of channels. Clear to maintain original channel numbering.
Export outliers as unsorted	Select to mark outliers (sort code 31 in a TTank data tank) as "Invalid" when exported to PLX. This allows Offline Sorter to properly recognize these events as outliers.
Data Scaling Multiplier	Enter the desired scaling multiplier to be applied on export. This is typically set to 1000000 to convert floating point uV signals into integer format.
Import Options	
Automatically import sorts	Select to allow auto-import of sorts when changes to PLX files in the working files folder are detected.
Confirm before importing	Select to prompt for import when changes to PLX files in the working files folder are detected.
Delete working files after import	Select to allow PLX files in the defined working files folder to be automatically deleted after new sort data are imported. This includes the PLX file with the new sort information.
Prompt for sort name	Select to prompt for sort name when sorts are imported.
Use fixed sort name	Select and define a sort name to be used automatically when sorts are imported.

Once all preferences have been set, ensure PLX is selected in the Export Format drop-down list.

Exporting Channels

Before exporting, set preferences and select data as described above. Ensure that the desired events, sorts and channel numbers are chosen in the Selector Panel

Click the *Export selected channels* button on the toolbar, or click "Export Channels" in the File I/O menu.

If the *Auto-run Offline Sorter (PLX / DDT)* option is selected in the OpenBridge Preferences, Offline Sorter will be launched and the exported data set will be loaded automatically. Otherwise, manually launch Offline Sorter and load the PLX file.

> **Note**
>
> The Offline Sorter Hardware key must be connected to enable all features. If a warning message is displayed, insert the key before continuing. You will need to close and re-open Offline Sorter.

The PLX file is stored in the working folder (set in the Preference dialog) and named using the following convention `TankName_BlockName_SnippetEventName__Number.plx` where `Number` is incremented on each subsequent export.

Exporting Tetrode Recordings

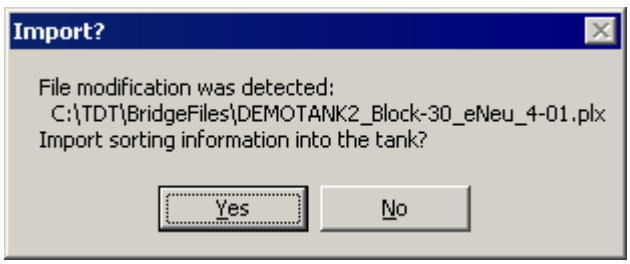
The Tetrode Sorting gizmo in Synapse concatenates the four snippets of a tetrode into one long snippet and saves that to a single channel in the data tank. When exporting to PLX, Plexon's Offline Sorter expects the data from each tetrode to be in four unique channels. OpenBridge automatically detects tetrode data in the OpenEx data tank and splits each tetrode snippet into individual channels on export. However, sort codes identified in Offline Sorter can not be transferred back into the TDT data tank at this time.

Importing Channels

After you have completed sorting and analysis in Offline Sorter, save the file to the OpenBridge Working directory.

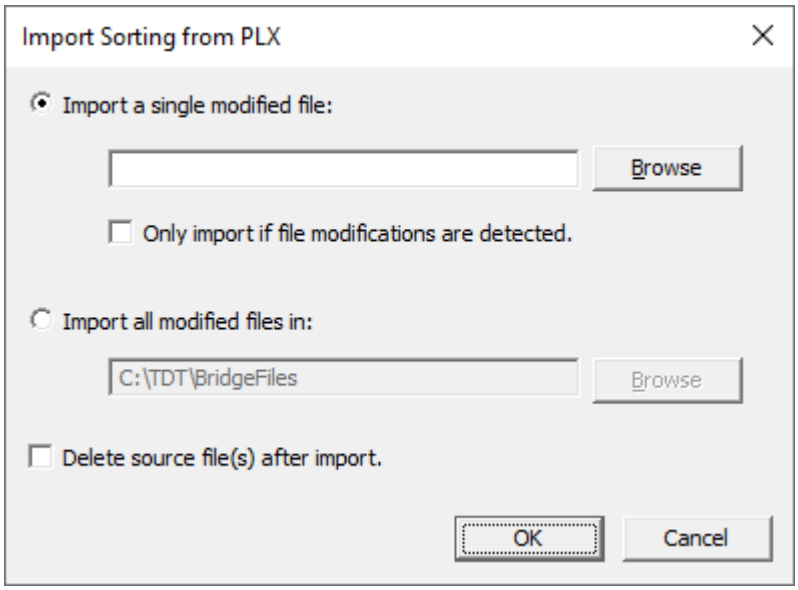
Depending on selections in the Preferences dialog, OpenBridge may:

- Auto-detect the new sort data
- Automatically import new sort data
- Prompt for confirmation before importing



- Prompt for a sort name before importing

If OpenBridge is not set to auto-detect new sort information, you can import the information manually by clicking the Import Sorted Files button, to launch the Import Sorting from PLX dialog.



In this dialog you can choose to import sort information from a single file or all files in the working directory. You can also choose whether or not to delete source file(s) after import.

1. Click the desired radio button and browse for the new file.
2. Click OK.
3. If you are prompted to enter a sort name, type the new name and click OK.

The Toolbar and Menus

The OpenBridge Toolbar

The Standard Toolbar provides access to the most common OpenBridge commands.

 Export Selected Channels	Export selected channels to the working files location in the selected export format
 Export Selected Channels in Chunks	Export selected channels in chunks to the working files location. This is only available when exporting to PLX format. Each chunk of channels is exported into a separate file. The number of channels in each chunk is set in the Preferences dialog PLX tab.
 Import Sorted Files	Launch the Import Sorting from PLX dialog, enabling users to select a single file or all modified PLX files for import
 Open Export Directory	Open the working files location in Windows Explorer
 Setup Preferences	Launch the Preferences dialog box

OpenBridge Configuration Menu

Import	Imports previously exported configurations information from an *.obconf file. Helps ensure consistent settings are used from block to block.
Export	Exports all necessary export configuration settings, including export format and preference settings, channels/sorts as well as the setting for the desired sample rate for stream exports.