OFS 4.1.0 Matlab Import Notes

Offline Sorter v4.1.0 introduces the ability to import continuous or spike waveform data from a Matlab .mat file. A frequently asked question about Offline Sorter in Support is how to get data from Matlab into Offline Sorter, and the answer we'd give was to use the NeuroExplorer SDK. This new feature allows users to save workspace variables inside Matlab to a .mat file and load directly into Offline Sorter.

Arranging data inside Matlab into arrays that Offline Sorter can understand can be a bit tricky. Furthermore, there are decisions to be made about the data scale, how to represent it in the .mat file, and a few rules to follow concerning the 32 and 64-bit versions of Matlab and Offline Sorter.

This document is meant to supplement the information in the updated Offline Sorter user guide.

Continuous and Spike Waveform Data

The Matlab data import allows you to import continuously digitized data, or spike waveform data, but not both at the same time.

Bit Depth and Matlab Installation Rules

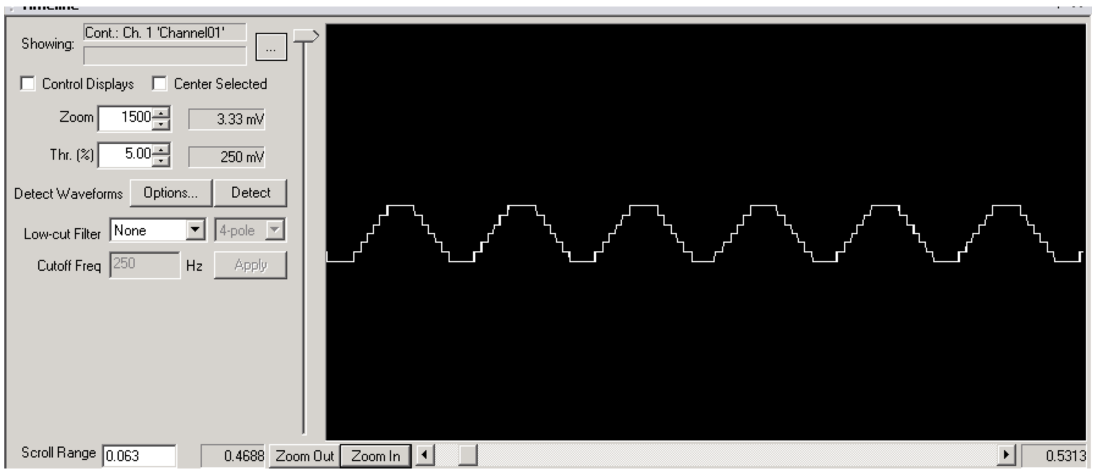
Offline Sorter 32-bit can only import data from a .mat file that was created using Matlab 32-bit. Offline Sorter 64-bit can only import data from a .mat file that was created using Matlab 64-bit. Offline Sorter uses .dlls installed by Matlab to load the .mat file, which means Matlab (of a matching bit-depth) must be installed on the same computer you're running Offline Sorter on.

Voltage or ADC Counts Scaling

Both the continuous data and spike waveform import have the option to import data that is represented in the .mat arrays as ADC counts or microvolts. One of the import parameters is "Maximum Voltage (mV)" of the A/D converter.

What's very important to understand is that the continuous or waveform values are always assumed to be from a 16-bit A/D converter, and you'll want to make sure the maximum voltage is set appropriately as to represent the imported data as accurately as possible.

One mistake I made while getting familiarized with this feature was taking the parameters too literally. I recorded a sine wave from a function generator into an OmniPlex system through the HTU. The voltage values at the headstage input were +/- 500 uV. I saved this data in microvolts to a .mat file and set the maximum voltage to 5000 mV (since it was into a +/- 5V A/D converter). The result was not very good.


Showing: Cont.: Ch. 1 'Channel01'

Control Displays ☐ Center Selected ☐

Zoom 1500 — 3.33 mV

Thr. (%) 5.00 — 250 mV

Low-cut Filter None — 4-pole

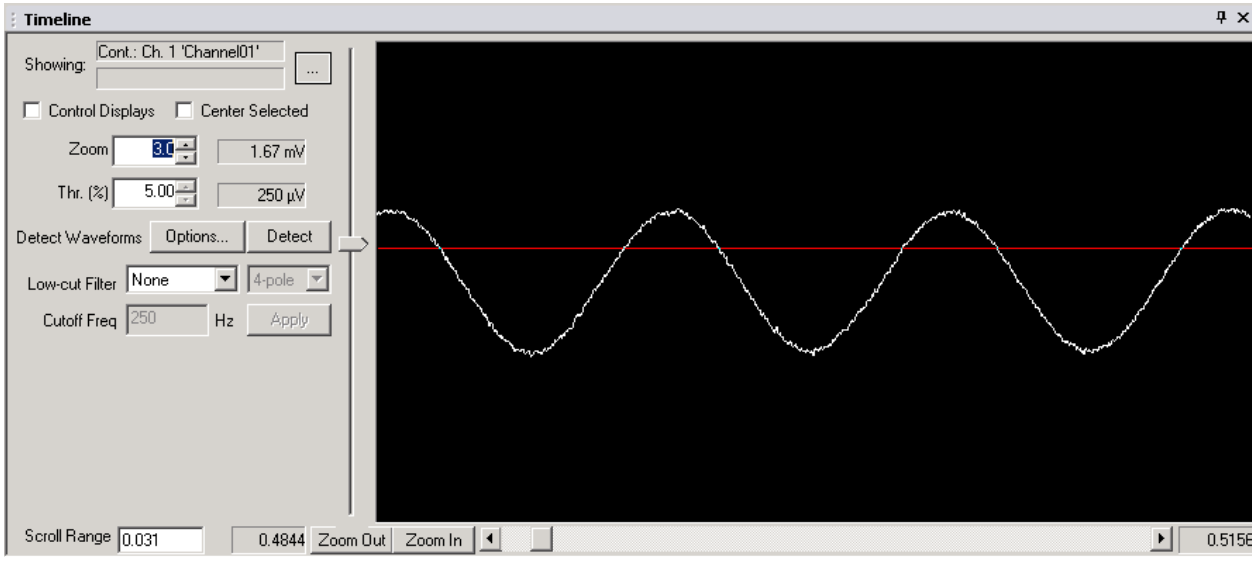
Cutoff Freq 250 Hz

Scroll Range 0.063 — 0.4688 … 0.5313

I had to zoom in 1500x to see the sine wave, and it was heavily quantized. The reason this happened is obvious in hindsight. OFS assumed that this data came from a 16-bit A/D converter. Since I set the max voltage to 5V, there are 153 microvolts per bit. My 500 uV sine wave is only represented by about 4 bits out of 32767!

There are three strategies (that I know of) for getting the best import results.

1a) Set the maximum voltage to the actual minimum and maximum values contained in the data, or alternatively,
1b) set the maximum voltage to the A/D converter range with the front-end hardware gain divided out. Since my front end hardware had a gain of 1000x, I set the maximum voltage to 5 mV and can see much better results.


Timeline

Showing: Cont.: Ch. 1 'Channel01'

Control Displays ☐ Center Selected ☐

Zoom 3.0 — 1.67 mV

Thr. (%) 5.00 — 250 µV

Low-cut Filter None — 4-pole

Cutoff Freq 250 Hz

Scroll Range 0.031 — 0.4844 … 0.5156

2) Multiply the front-end gain into the data in Matlab, so that the voltage values in the .mat file are what the A/D converter saw, and set the maximum voltage in the import options to the range of the A/D converter. You can then tell OFS after import what the total gain on the channels are (Tools->Adjust Gain) so that the voltage values represent what was at the front of the headstage.

3) Similar to #2, save raw AD samples to the .mat file, and correct for the gain through Adjust Gain.

Continuous Import Example

Here's an example of how to get continuous data into the correct array type and arrangement that Offline Sorter expects. This example is a bit contrived since the data is first pulled from a .pl2 file through the Plexon SDK, but the concepts still apply.

First I'll get four channels of wideband from 4chDemoPL2.pl2 into four separate variables (WB01 through WB04):

```
>> [~, ~, ~, ~, WB01] = plx_ad_v('4chDemoPL2.pl2', 'WB01');
>> [~, ~, ~, ~, WB02] = plx_ad_v('4chDemoPL2.pl2', 'WB02');
>> [~, ~, ~, ~, WB03] = plx_ad_v('4chDemoPL2.pl2', 'WB03');
>> [~, ~, ~, ~, WB04] = plx_ad_v('4chDemoPL2.pl2', 'WB04');
```

The plx_ad_v.m function returns other values (such has number of points, fragment timestamps, etc), but I've chosen to ignore them since they're not important for this example. The only data kept from each channel is the actual voltage values in millivolts.

OFS expects the array containing the continuous data to be in the form array[channel][sample]. Each row is a channel, and the samples populate the columns of the respective rows.

plx_ad_v.m returns the voltage values in one long column, so we'll have to transpose each channel of data when adding it to the array we'll be loading into OFS.

Here's an example of how combining arrays work, if you want to append an array as a new row in an existing array:

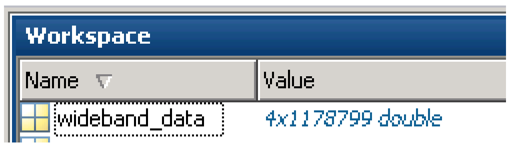
```
>> some_array = [1,2,3];
>> another_array = [4,5,6];
>> combined_array = [some_array; another_array]

combined_array =

    1    2    3
    4    5    6
```

The trick is that you use a semicolon, instead of a comma. This appends the data as a new row instead of starting with the next column.

So, to get all of our wideband data into four rows of one array…

```
>> wideband_data = [WB01'; WB02'; WB03'; WB04'];
```

The apostrophe after the individual channels of data transposes the array from one long column into one long row. If your data is already in a row, you won't have to do that. The workspace list in the Matlab UI can give you a good indication if you were successful. It shows for me that I have four rows of data with 1,178,799 voltage values along the columns.



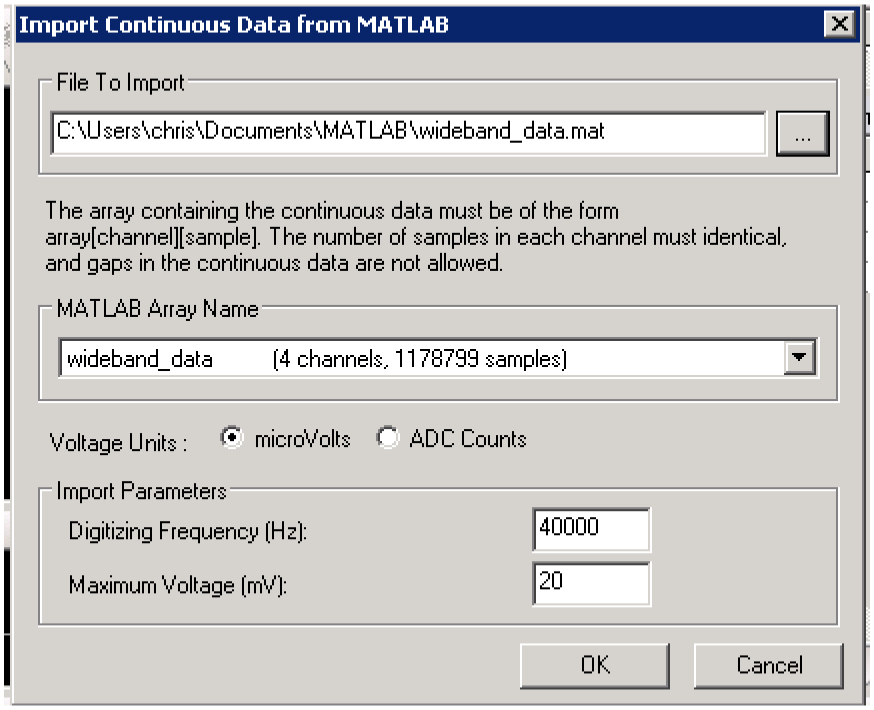
wideband_data 4x1178799 double

One last thing – our data is in millivolts, and OFS expects microvolts:

```
>> wideband_data = 1000 * wideband_data;
```

You can now right-click on the wideband_data variable in the workspace and click "save as" and save it as a .mat file.

To load this in Offline Sorter, go to "File->Import Continuously Digitized Data from Matlab", and select the .mat file that was saved. The array wideband_data will show up in the "MATLAB Array Name" drop-down.



File To Import: C:\Users\chris\Documents\MATLAB\wideband_data.mat

The array containing the continuous data must be of the form array[channel][sample]. The number of samples in each channel must identical, and gaps in the continuous data are not allowed.

MATLAB Array Name: wideband_data (4 channels, 1178799 samples)

Voltage Units : [✓ microVolts] ADC Counts

Import Parameters

Digitizing Frequency (Hz): 40000

Maximum Voltage (mV): 20

Our data is in microvolts, so choose that option. The digitization rate is 40 kHz, and the max voltage (taking a gain of 250x and a 5V max A/D range into account) is 20 mV.

Once imported, it can be thresholded/sorted and saved as .plx or .nex.

<u>Spike Waveform Import Example</u>

Here's an example of how to get spike waveform data into the correct array type and arrangement that Offline Sorter expects.

First I'll get four channels of spike waveform data from 4chDemoPL2.pl2 into four separate variables (SPK01 through SPK04) with the corresponding timestamps of each spike (TS1 through TS4):

```
>> [~, ~, TS1, SPK01] = plx_waves_v('4chDemoPL2-unsorted.pl2', 'SPK01', 0);
>> [~, ~, TS2, SPK02] = plx_waves_v('4chDemoPL2-unsorted.pl2', 'SPK02', 0);
>> [~, ~, TS3, SPK03] = plx_waves_v('4chDemoPL2-unsorted.pl2', 'SPK03', 0);
>> [~, ~, TS4, SPK04] = plx_waves_v('4chDemoPL2-unsorted.pl2', 'SPK04', 0);
```

The plx_waves_v.m function returns other values, but I've chosen to ignore them since they're not important for this example. The only data kept from each channel are the waveforms in millivolts, and the timestamps of each waveform in seconds.

OFS expects the waveform data to be in a Matlab type called a cell array. Cell arrays are able to hold non-homogenous data, which is important since each spike channel is pretty much guaranteed to have different numbers of waveforms. Each cell in the cell array must contain an array of one channel of spike waveforms in the form array[spike][sample]. Each row is a spike, and the samples of the waveform populate the columns.

Here's an example of adding matrix arrays to a cell array. Take note of the curly braces used in assigning data to an element of a cell array.

```
>> cell_array = {an_array; another_array}
```

plx_waves_v.m returns the waveform data already in the expected array[spike][samples] arrangement. It also returns the data in millivolts, so you'll have to multiply each returned waveform array by 1000.

```
>> SPK01 = SPK01 * 1000;
>> SPK02 = SPK02 * 1000;
>> SPK03 = SPK03 * 1000;
>> SPK04 = SPK04 * 1000;
```

To get the spike data in to a cell array…

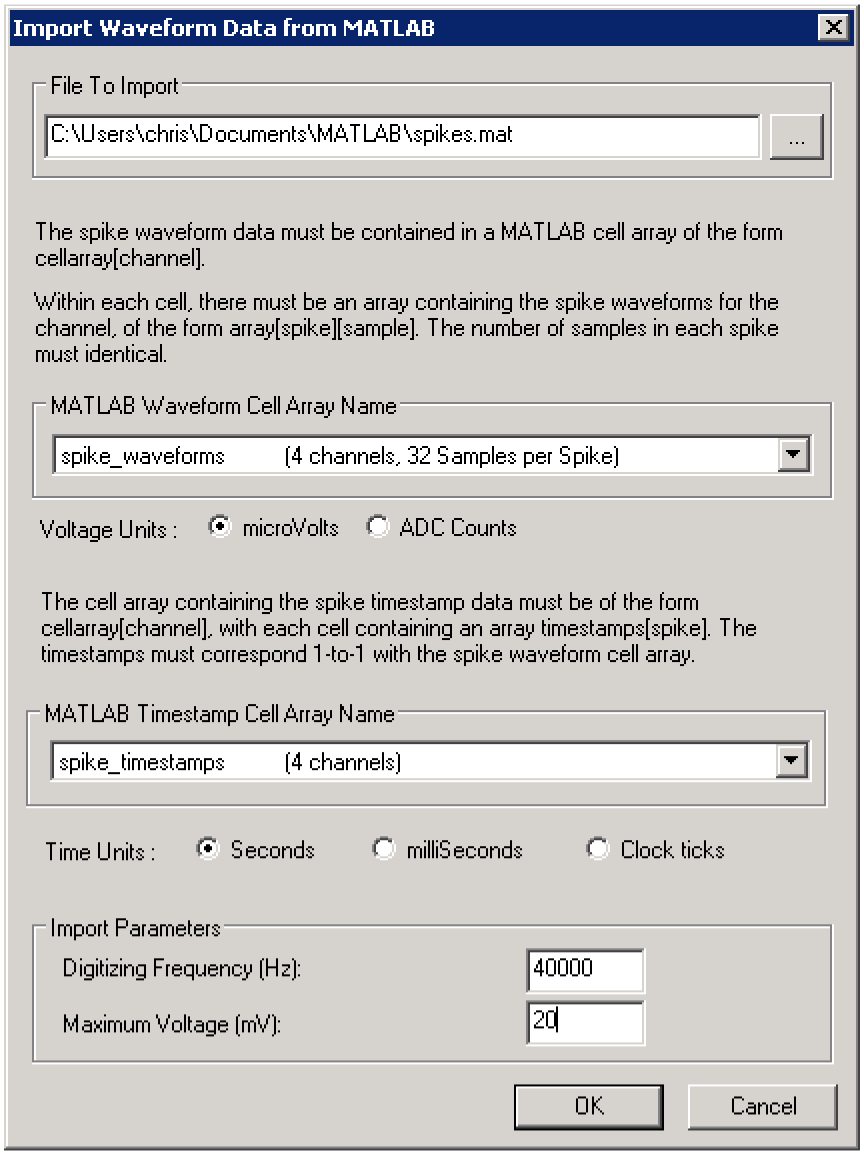
```
>> spike_waveforms = {SPK01; SPK02; SPK03; SPK04};
```

OFS also needs the timestamps in a cell array in the form cellarray[channel], which each cell containing an array timestamps[spike]. The timestamps must correspond 1-to-1 with the spike waveform cell array.

```
>> spike_timestamps = {TS1; TS2; TS3; TS4};
```

Select both spike_waveforms and spike_timestamps in the workspace list, right-click, and select 'save-as' to save to a .mat file.

To load this in Offline Sorter, go to File->Import->Spike Data from Matlab, and select the .mat file that was saved. The array spike_waveforms will show up in the "MATLAB Waveform Cell Array Name" drop-down, and the array spike_timestamps will show up in the "Matlab Timestamp Cell Array Name" drop-down.


Import Waveform Data from MATLAB

File To Import: C:\Users\chris\Documents\MATLAB\spikes.mat

The spike waveform data must be contained in a MATLAB cell array of the form cellarray[channel].

Within each cell, there must be an array containing the spike waveforms for the channel, of the form array[spike][sample]. The number of samples in each spike must identical.

MATLAB Waveform Cell Array Name: spike_waveforms (4 channels, 32 Samples per Spike)

Voltage Units : [✓ microVolts] ADC Counts

The cell array containing the spike timestamp data must be of the form cellarray[channel], with each cell containing an array timestamps[spike]. The timestamps must correspond 1-to-1 with the spike waveform cell array.

MATLAB Timestamp Cell Array Name: spike_timestamps (4 channels)

Time Units : [✓ Seconds] milliSeconds Clock ticks

Import Parameters

Digitizing Frequency (Hz):	40000
Maximum Voltage (mV):	20

Our waveform data is in microvolts, and the timestamps are in seconds, so choose those options. The digitization rate is 40 kHz, and the max voltage (taking a gain of 250x and a 5V max A/D range into account) is 20 mV.

Once imported, it can be sorted and saved as .plx or .nex.