Documentation

fpExplorer

Contents

1. Description

This application should be used to preview and perform basic analysis of fiber photometry data. The application was designed for data collected by TDT system but can also handle data saved in csv files as long as they follow the correct layout (see 2.2).

2. Quick guide

2.1. Load TDT Data

fpExplorer.exe file should be used to start the application. Then, the users will be presented with the main application window where they can load TDT data by clicking on “Select TDT Data” button. (Fig1)

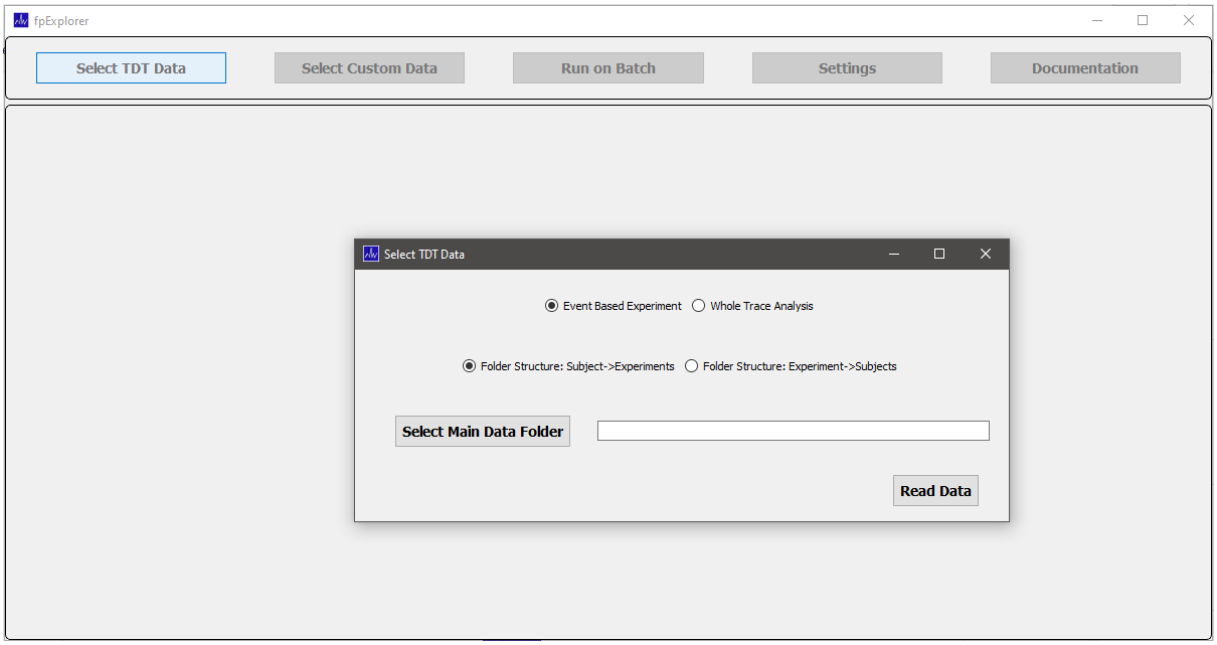


Fig1. Application main window with “Select Data” window where user defines what kind of data will be analyzed.

2.1.1. Event Based Experiment

If the dataset contains recorded events that don’t start with Cam or Tick(i.e., PrtA 253, PrtA 249), the user can analyze it as event based experiment. In this case, the user will be able to preview raw data, trim, downsample and normalize the data, and detect spikes. Most importantly, the user will be able to plot events and to perform peri-event analysis.

2.1.2. Whole Trace Analysis

If the dataset does not contain recorded events that don’t start with Cam or Tick (i.e., PrtA 253, PrtA 249), the user can analyze it as the whole trace. In this case the user will be able to preview raw data, trim, downsample and normalize the data, and detect spikes.

2.1.3. Folder Structure

The application is designed to analyze two kinds of data structures from TDT system. (https://www.tdt.com/docs/synapse/managing-data-for-your-lab/)

Subject -> Experiments

A folder that contains subfolders with subject names. Within subject subfolders, there should be subfolders with experiment names. Within experiment subfolders, there should be user's TDT files (i.e., *.Tbk, *.Tdx, *.tev, *.tin, *.tnt, *.tsq).(Fig2)

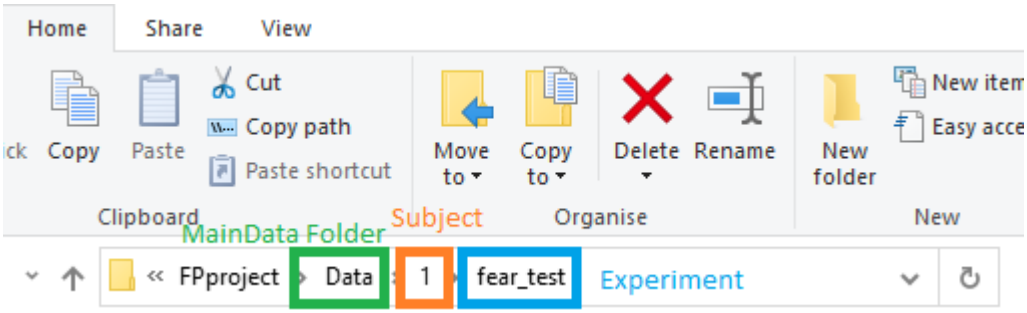


Fig2. Subject->Experiment folder structure example, where fear_test is an experiment name with TDT files (i.e., *.Tbk, *.Tdx, *.tev, *.tin, *.tnt, *.tsq)

Experiment -> Subjects

A folder that contains subfolders with experiment names. Within experiment subfolders, there should be subfolders with subject names. Within subject subfolders, there should be user's TDT files (i.e., *.Tbk, *.Tdx, *.tev, *.tin, *.tnt, *.tsq).(Fig3)

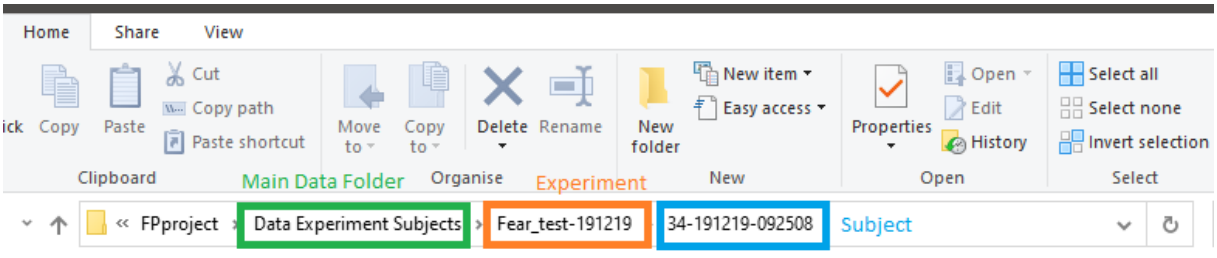


Fig3. Experiment->Subject folder structure example, where 34-191219-092508 is a subject name with TDT files (i.e., *.Tbk, *.Tdx, *.tev, *.tin, *.tnt, *.tsq)

2.1.4. Select Experiment Name

The user is prompted with the experiment names available from the first data set in the main folder. (Fig4)

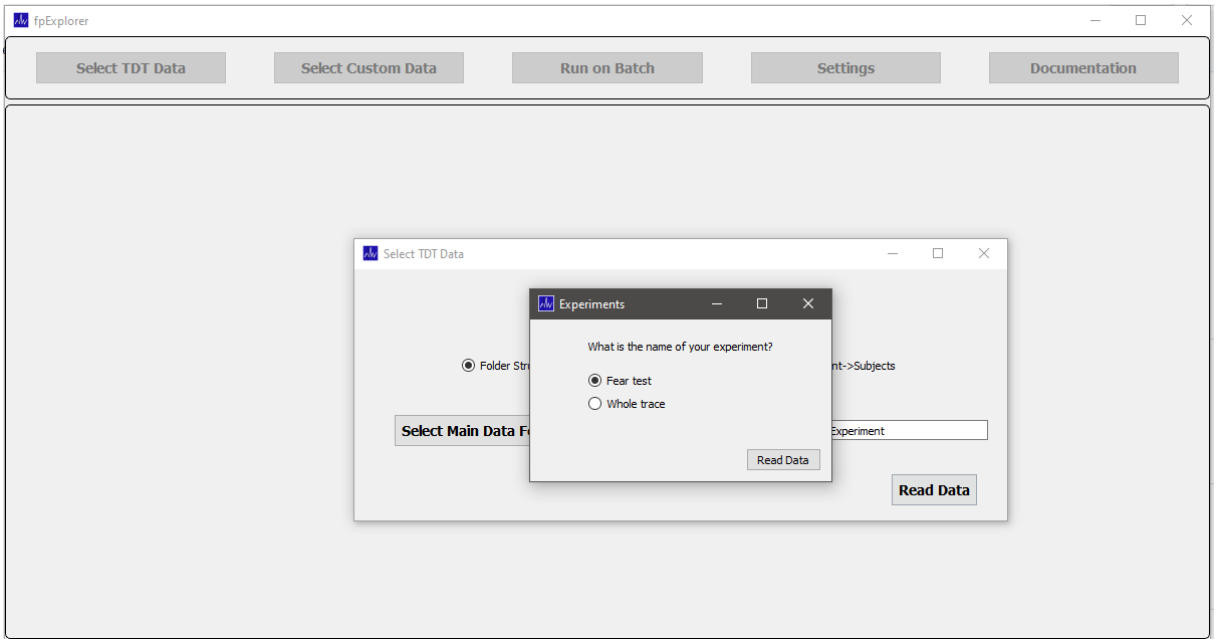


Fig4. Select experiment that will be analyzed

2.1.5. Select Signal and Control Channels

The user is prompted with channel names available from the first data set in the main folder. The user should select signal channel and control channel (Fig5)

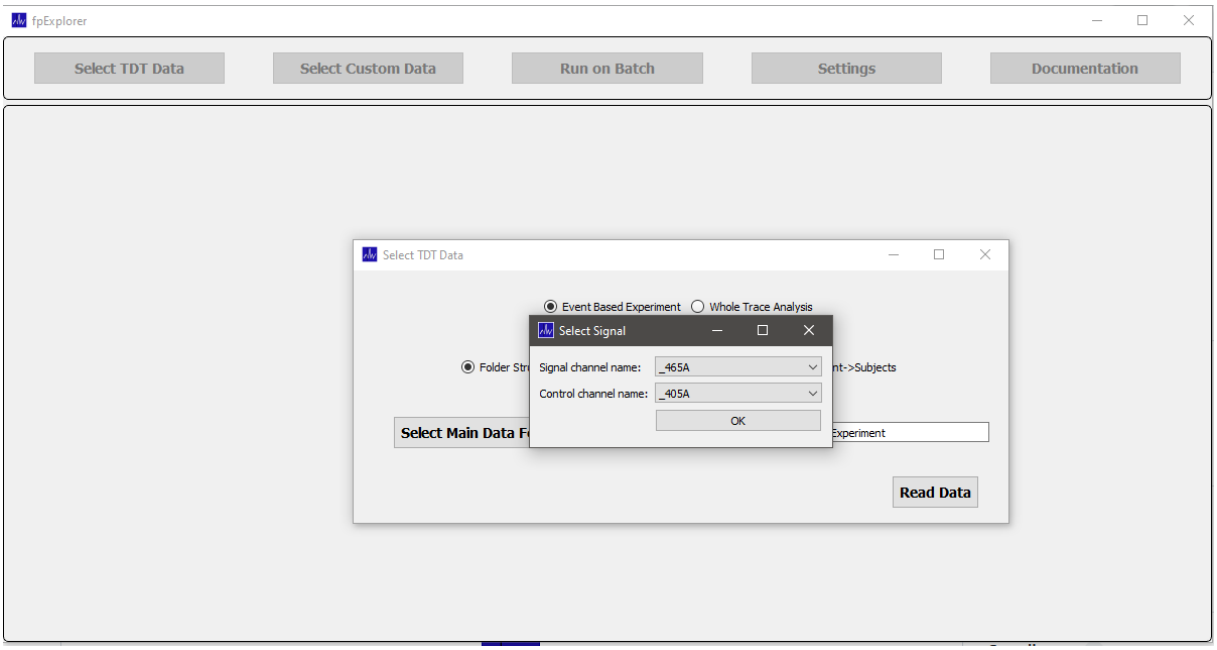


Fig5. Select signal channel name and control channel name.

2.2. Load Custom Data

The users can also load data from custom csv file. Clicking on “Select Custom Data” button will allow to enter the paths to user’s custom data csv file and the event csv file. (Fig6)

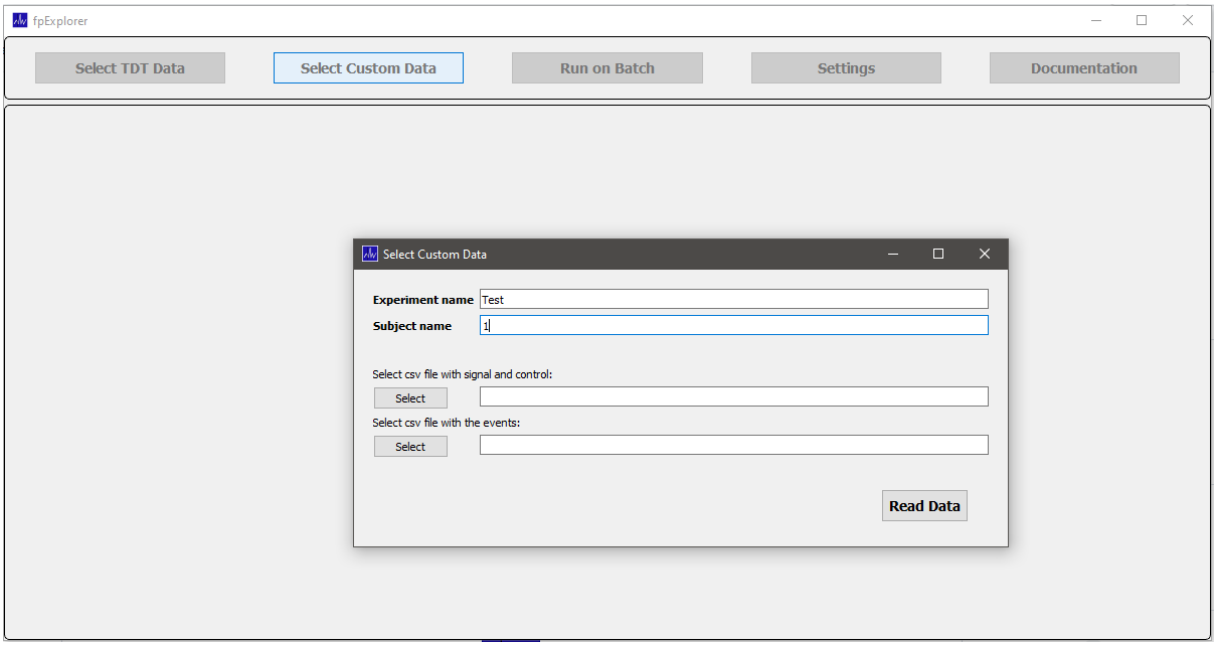


Fig6. Enter paths for custom data .csv and event .csv files.

IMPORTANT NOTE! The application only supports csv files with data organized in a specific way. For the data file, the first column must consist of time stamps (in seconds). The frequency will be later calculated based on those time stamps. The second column must consist of signal data. And the third column must consist of control data. The names of the headers in csv file can be different than the ones in the example, however the order needs to match the example. (Fig7)

	A	B	C
1	TimeStam	Signal	Control
2	1.96608	580.9991	91.15752
3	1.967063	581.0179	91.19018
4	1.968046	581.0346	91.22137
5	1.969029	581.0482	91.2509
6	1.970012	581.0591	91.27853

Fig7. The example csv file with signal and control data.

If the user also has a separate csv file with the events, they can preview those together with the recording. That csv file then also needs to have a specific structure. The first column should consist of event names (there can be multiple events in the same file). The second column should have event onset times (in seconds). Optionally, the third column could contain event offset times (in seconds) in case we add features in later versions of fpExplorer that depend on that data. The names of the headers don't have to be exactly as in the example. (Fig8). It is important, however, that the event onset times match as closely as possible the time stamps in the csv file with photometry data.

	A	B	C
1	Event	Onset	Offset
2	PAB_16	160.7157	165.7142
3	PAB_2064	165.7142	165.8157
4	PAB_0	165.8157	210.8159
5	PAB_16	210.8159	215.8143
6	PAB_2064	215.8143	215.9159

Fig8. The example csv file with event onset and offset times.

2.3. Preview

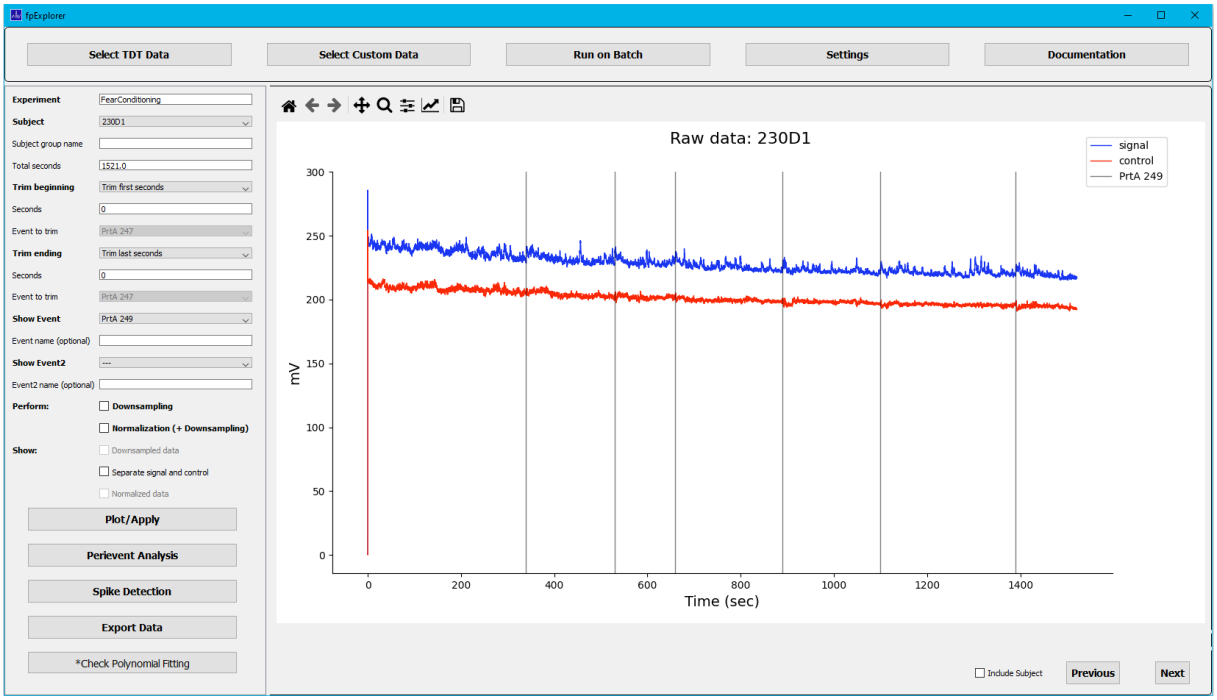


Fig9. The example preview of raw data read from TDT data files. Main options for subject based data analysis are visible to the left of the plot.

2.3.1. Options

Subject

The user can select any subject from the drop-down menu or use “Next” and “Previous” buttons to move between different subjects.

Subject group name

Here, the user can assign a group name to the subject.

Trimming

The user can decide to remove certain number of seconds from the beginning or ending of the recording before the analysis. The user can also opt to trim beginning or ending based on the first or last onset of the recorded event.

Events

The user can visualize up to two events available from the drop-down menu. It is also possible to enter a custom name for each selected event.

Downsampling and Normalization

The user can choose to downsample the signal and to normalize (to fit the control to the signal in order to correct for e.g. photobleaching and motion artifacts).

By clicking on the “Plot/Apply” button, the user can visualize the data. (Fig10)

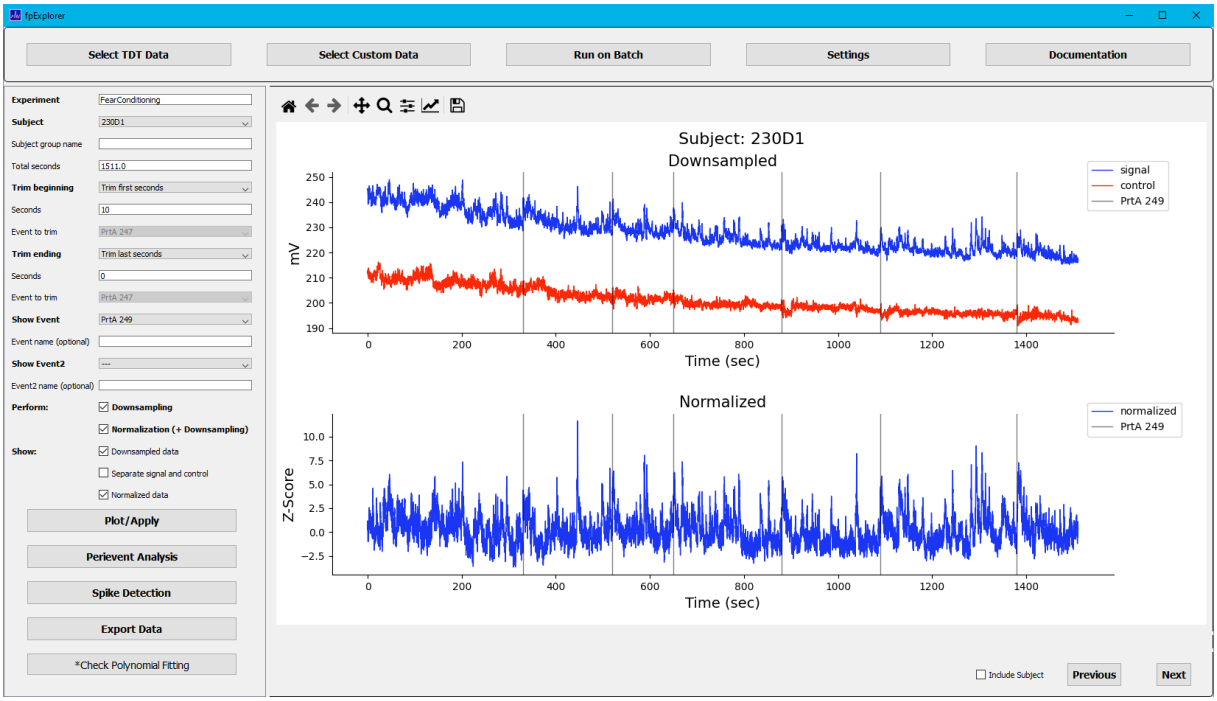


Fig10. The example of data visualization.

In order to check how well the polynomial fit worked, the user can click on ‘*Check Polynomial Fitting’ button. (Fig11)

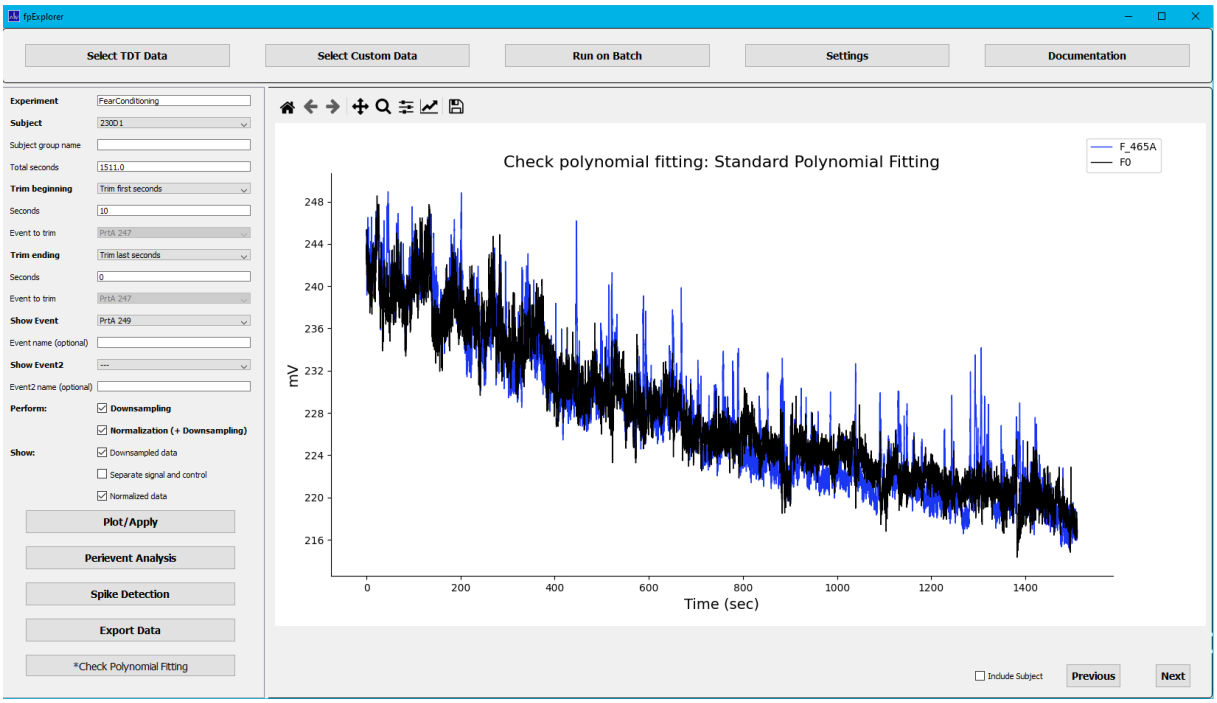


Fig11. Option to visualize how well the polynomial fitting works.

2.3.2. Settings

Downsample

The user can modify the sampling rate or normalization method at any time by clicking on "Settings" button from the main menu at the top of main application window. (Fig12)

We recommend downsampling the data to make its processing in fpExplorer faster as well as to make it possible to open exported data from csv files in spreadsheet software. The recommended downsampled sampling rate will depend on the biosensor that is being used, but 20-50Hz should generally prevent loss of relevant information.

Normalize

At the moment, the application allows to normalize data using either the default Standard Polynomial Fitting (least-squares linear fit method commonly used in the field and based on e.g. Lerner et al. 2015, Cell 162) or Modified Polynomial Fitting (a least-squares linear fit method adapted from Dr. Patrick J. Mulholland and described in e.g. Braunscheidel et al. 2019, J Neuroscience 39(46)). These methods will be described later. Then the user has the option to either show normalized data as df/F in % or as a Z-score.

Filter

The user can also choose to smooth data using a filter with a window around each sample of the data between 0 and 100000 samples (10 samples is the default).

We are using a zero-phase filter, i.e. a digital filter forward and backward to the signal from Python's scipy.signal package called filtfilt:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.filtfilt.html

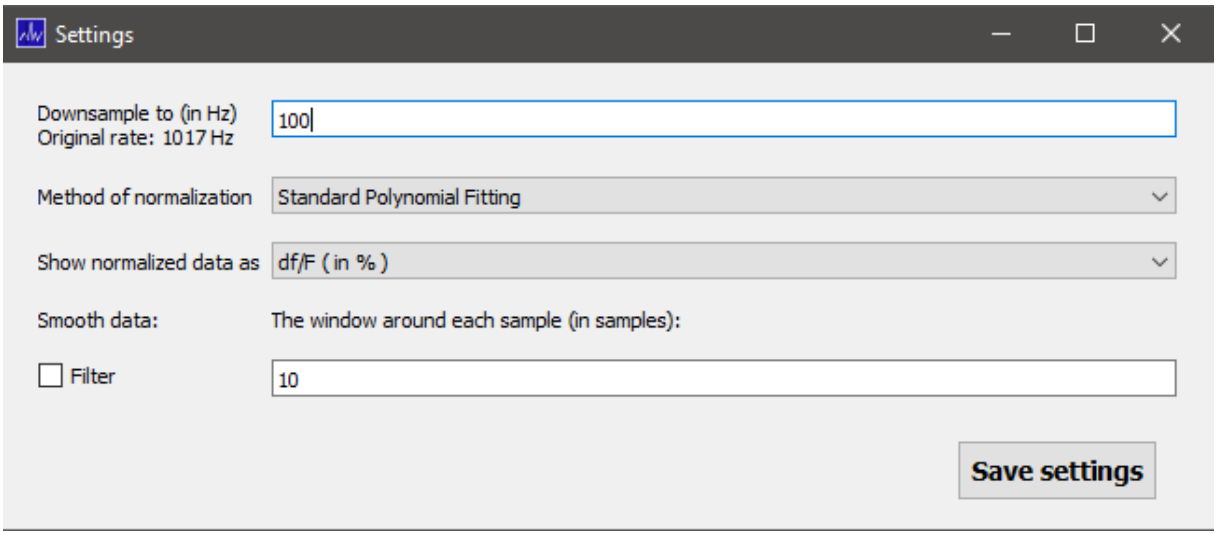


Fig12. Settings window where the user can change normalization method and downsampling applied during the analysis.

2.3.3. Plots

Zoom

The user can zoom in on the data by scrolling the mouse (zooms only along x axis) or by clicking on a magnifying glass and drawing a rectangle over a place of interest. Left and right arrows go back and forward between rectangle zooming.

Move

The user can move the plots by selecting a cross and then by dragging and dropping plots.

Adjust Plot Layout

The user can modify the spaces between subplots by clicking on the icon with three sliders. (Fig13)

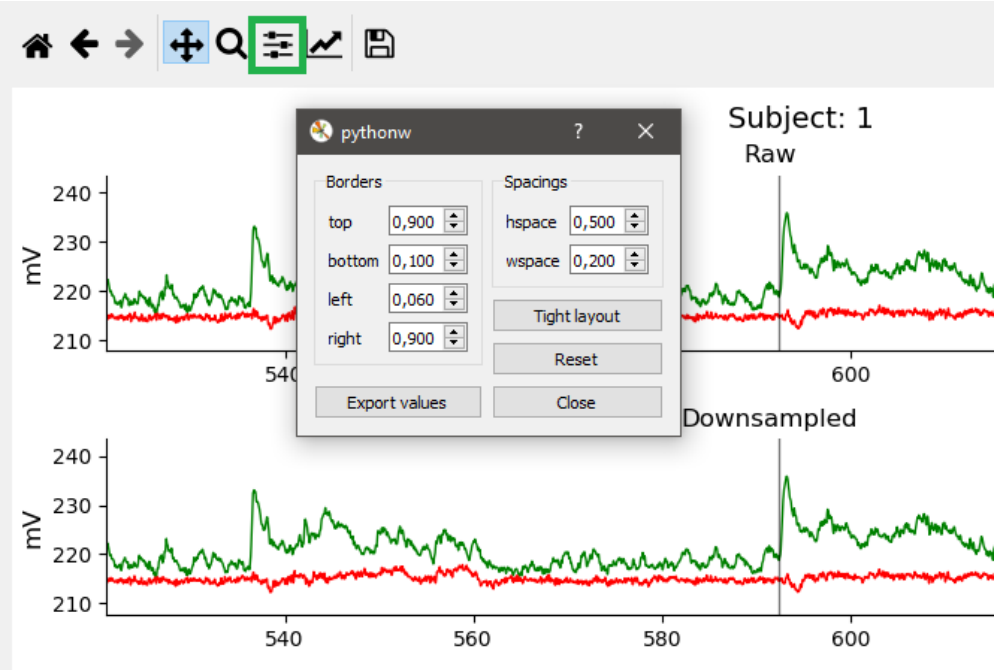


Fig13. Option to adjust subplots layout.

Adjust Labels, Range or color

The user can also customize the plot's title, axis labels and displayed data range. (Fig14)

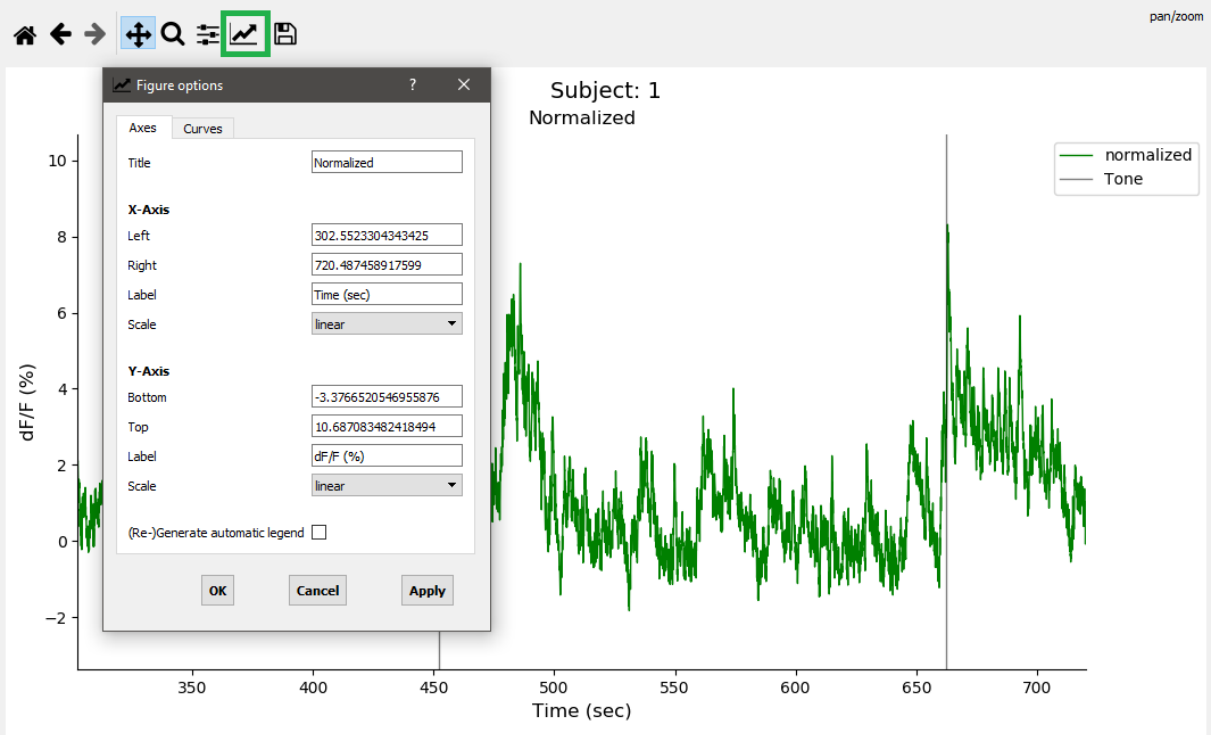


Fig14. Option to customize the plot.

Save

By clicking on a disk icon, the user can save a custom plot at any time.

2.4. Analysis

2.4.1. Perievent Analysis

By clicking on the “Perievent Analysis” button, the user can choose to perform some basic analyses around a selected event type throughout the recording (Fig15). After the first perievent preview, the user will be presented with the option to include or exclude single trials/instances of an event (Fig16).

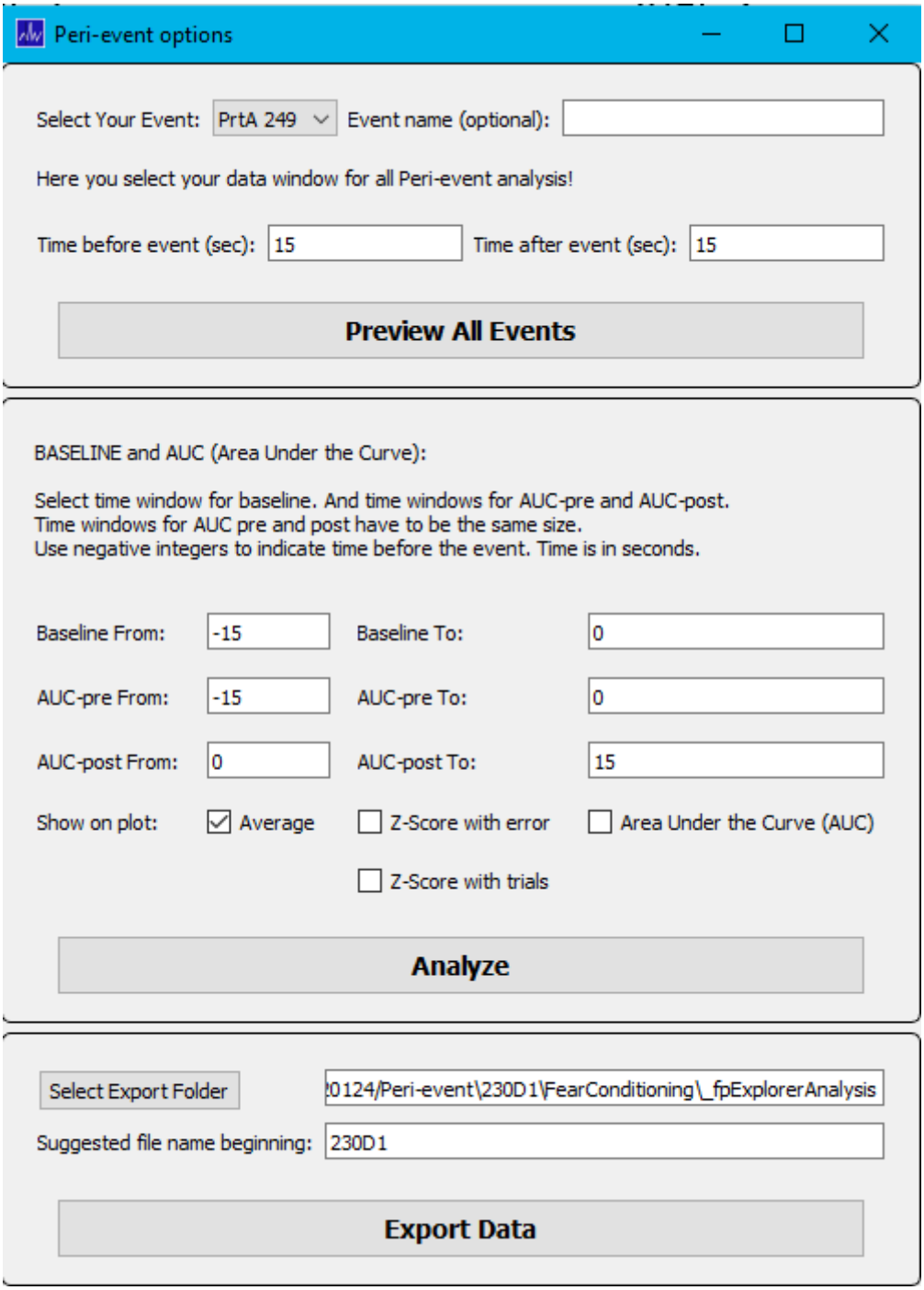


Fig15. Perievent analysis options.

Preview All Events

Here, the user can visualize normalized data around each instance of a selected event type throughout the recording. First, the user needs to select the event from the drop-down menu and enter how many

seconds before and after each event instance to show. This data window will also be used for the next peri-event data analysis. Optionally, the user can enter a custom event name. (Fig13)

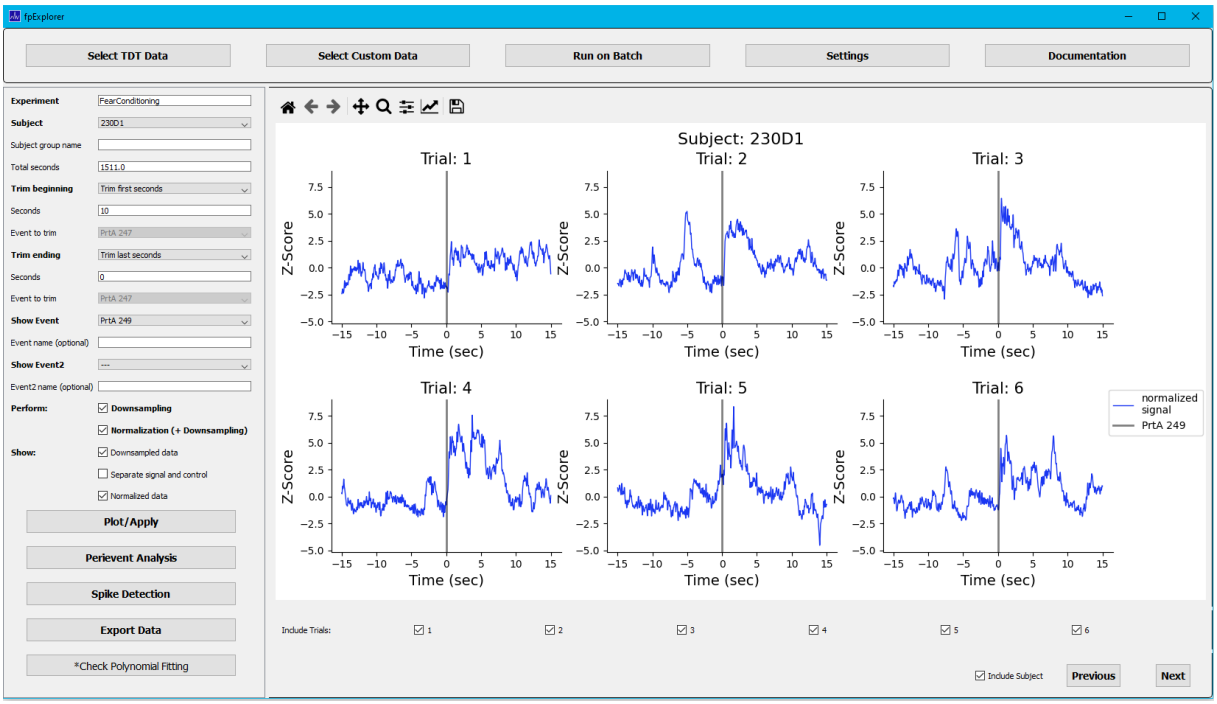


Fig16. Example of perievent preview.

Analyze

- Average

Here, the user can visualize average downsampled data around the selected events throughout the recording. This is the data before normalization, but after applying a vertical shift to center all data around y=0. First, the user needs to select the event from the drop-down menu and enter how many seconds before and after the event to show. Optionally, the user can enter a custom event name. (Fig17)

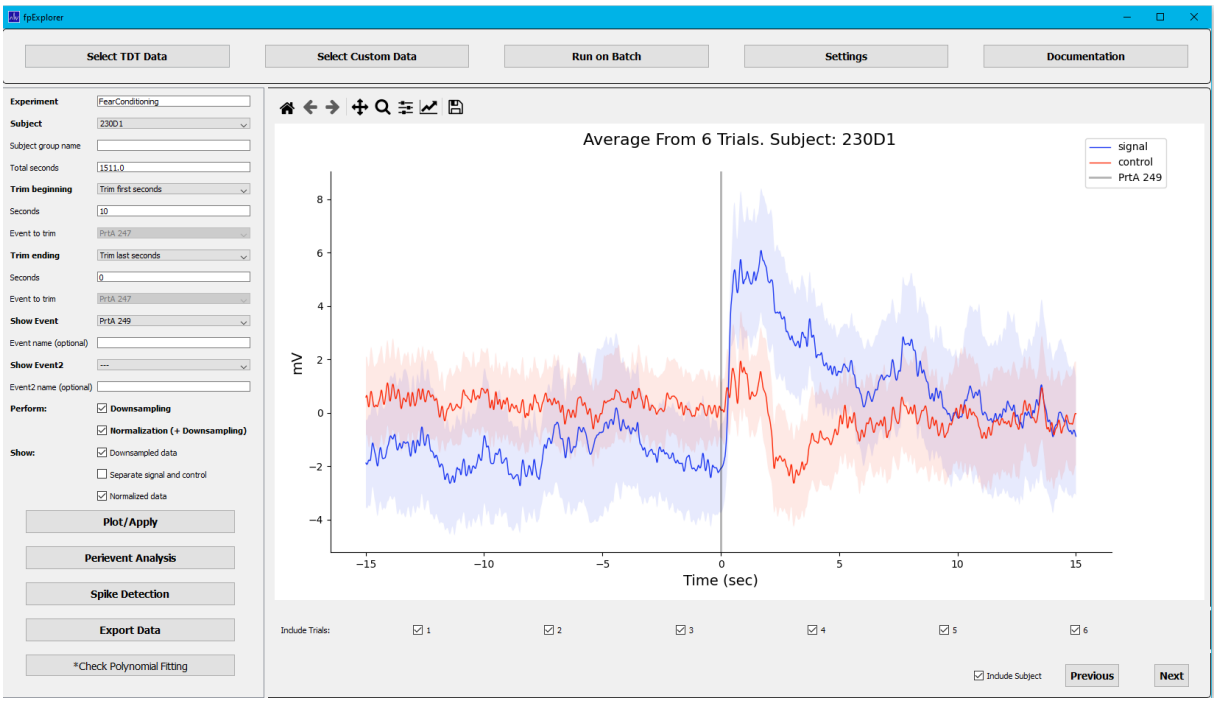


Fig17. Example of average perievent analysis.

- Z-score

Here, the user can visualize individual z-score traces around the selected event throughout the recording. First, the user needs to select the event from the drop-down menu and enter how many seconds before and after the event to show. Optionally, the user can enter a custom event name. Then, the user needs to define the time window for baseline (in seconds). The baseline must be within the selected data window. Negative integers must be used to indicate time before the event. Data within the baseline window will be used to calculate median and median absolute deviation (MAD) required for calculating z-score. (Fig18)

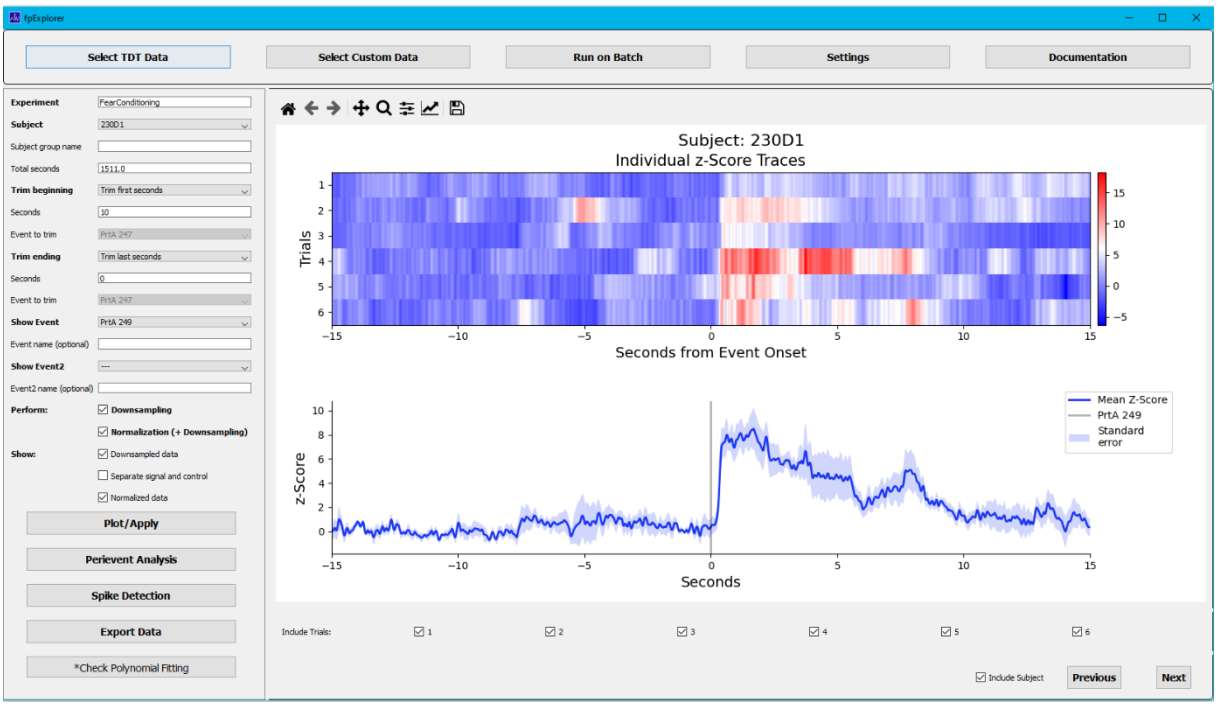


Fig18. Example of z-score visualization

- Area Under the Curve (AUC)

Here, the user can visualize Area Under the Curve as a bar chart of before and after the event. First, the user needs to select the event from the drop-down menu and enter how many seconds before and after the event. Optionally, the user can enter a custom event name. Then, user needs to define the time window in seconds for the “Pre event” and “Post event” periods. Negative integers must be used to indicate time before the event. The windows “Pre” and “Post” must be the same size. (Fig19)

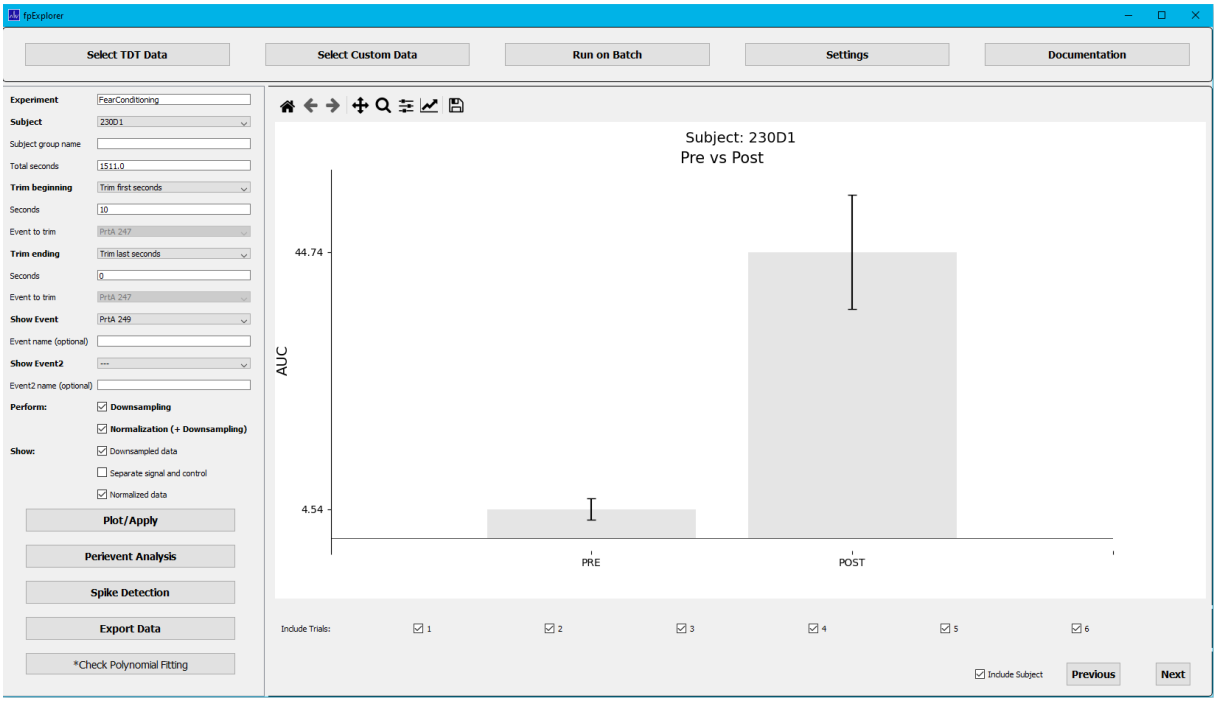


Fig19. The example of Area Under the Curve analysis.

Export Data

Here, the user can decide to export data for all of the above analyses. The user also has the option to save these plots as both png and svg files(Fig20)

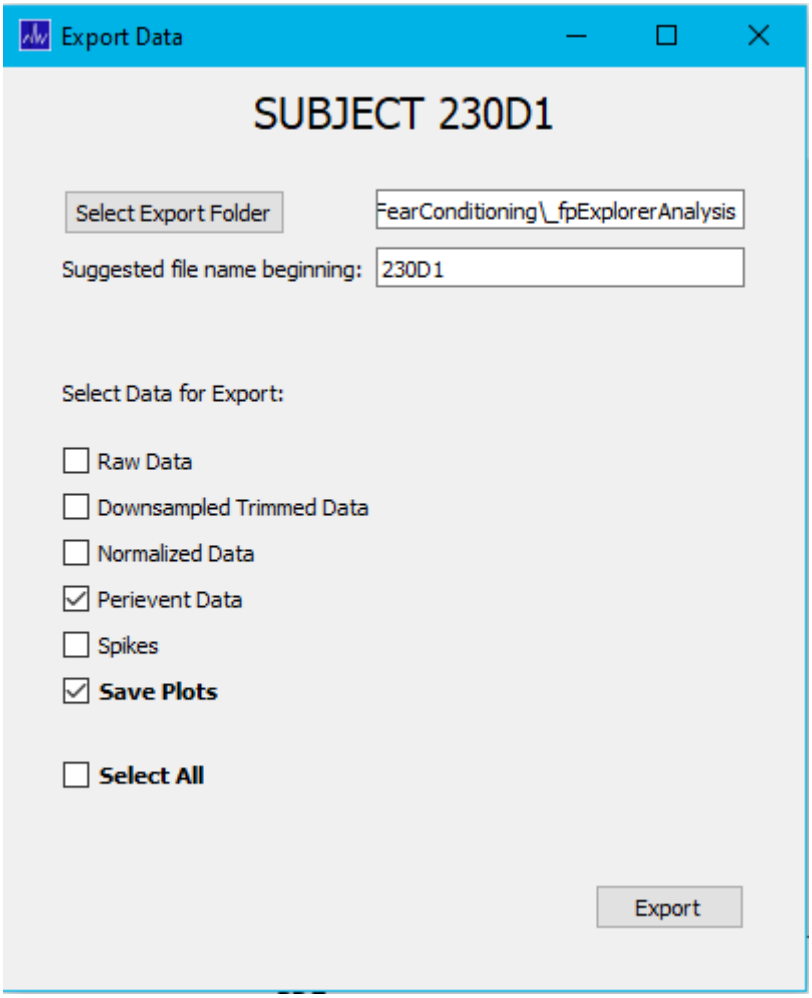


Fig20. Export Data window, from where the user can export both plots and data from the perievent analysis.

2.4.2. Spike Detection

By clicking on the "Spike Detection" button, the user can define different parameters that will allow to detect potential spikes or transients in the signal. (Fig21) The results of the spike detecting algorithm (from python's numpy package) will be displayed as a graph so the user can quickly verify if the parameters reliably detect what the user considers spikes. (Fig 22) The user is allowed to define up to 3 time windows in which fpExplorer will quantify the frequency and amplitude of detected spikes.

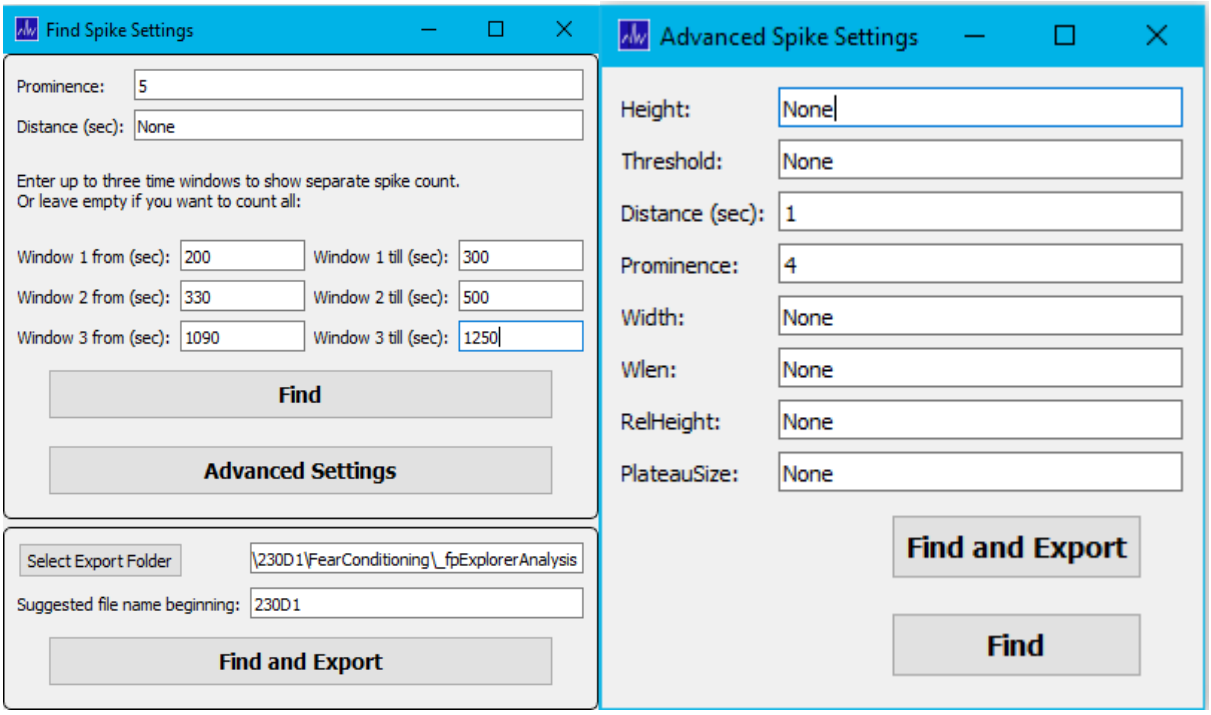
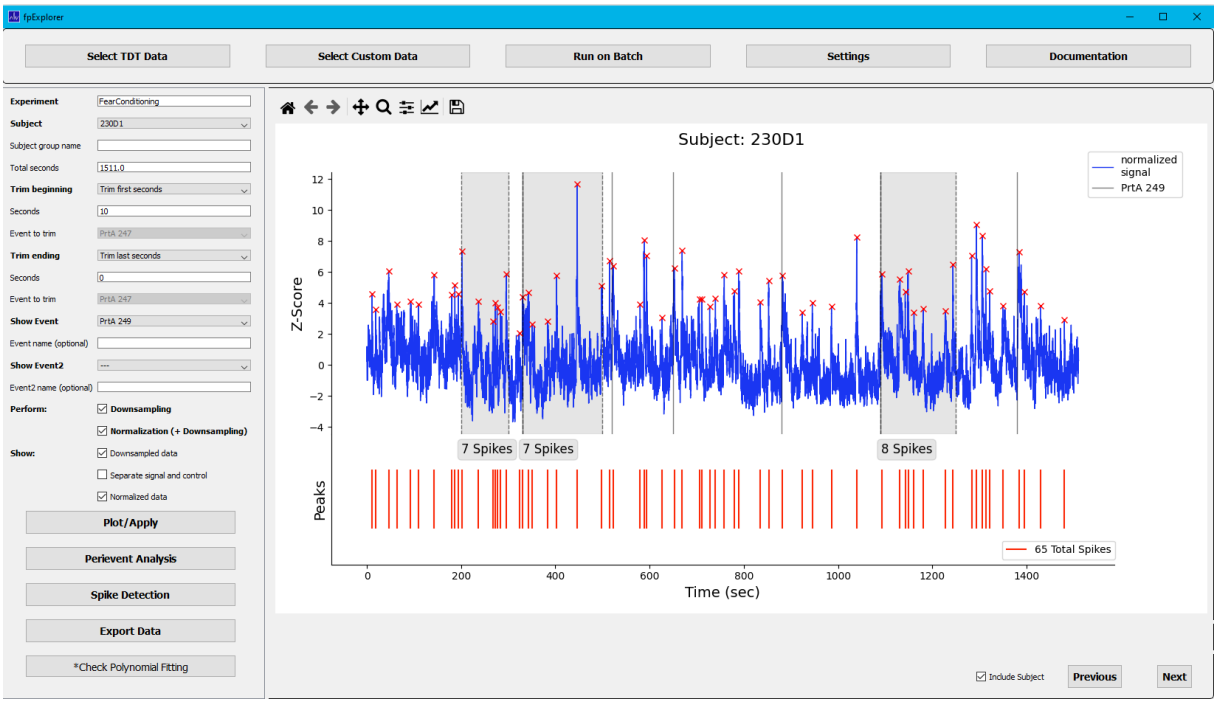


Fig21. Possible parameters that can help detect spikes withing the recording.



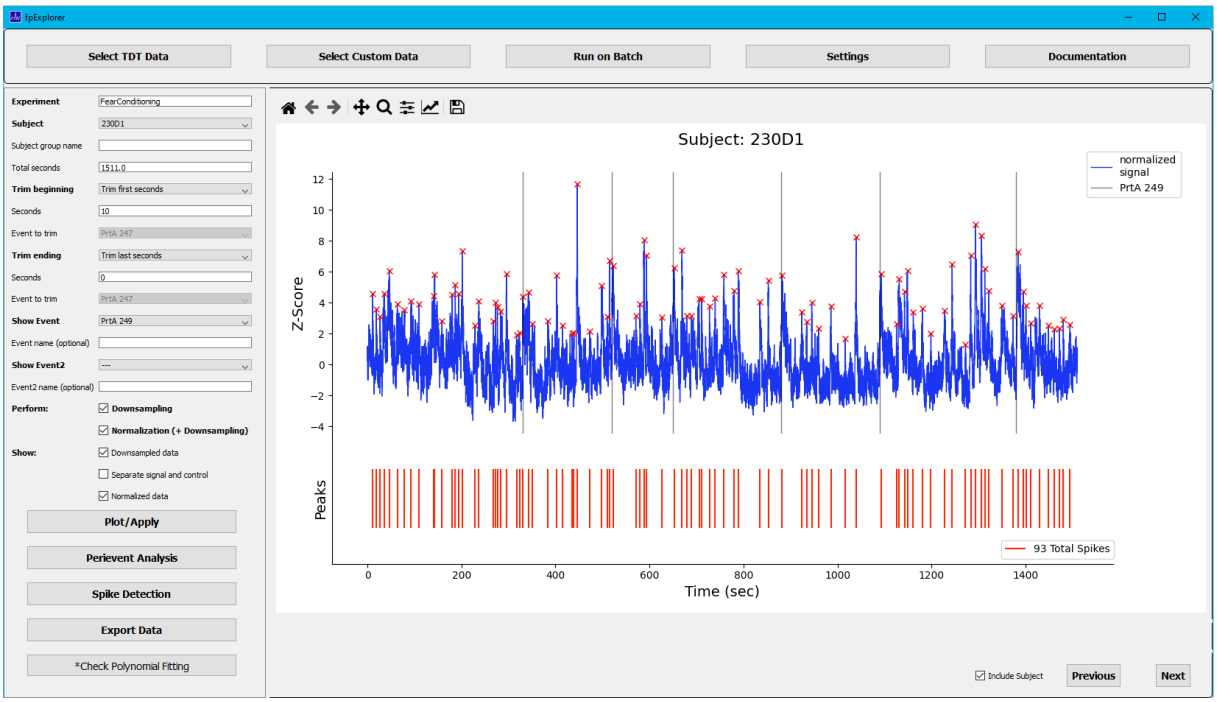


Fig22. Example results of spike detecting algorithm with user defined parameters.

2.5. Export Data

By clicking on “Export Data” button, the user can choose to save all data as csv files as well as the corresponding plots (as png and svg). This option only allows to export one selected subject’s data at a time. (Fig20) By default the application will recommend saving that data in the subfolder of each subject called “_fpExplorerAnalysis”. But this can be changed by the user to any custom folder.

2.6. Run on Batch

The “Run on Batch” button in the main application window opens a window with options that allow user to run automated analysis on multiple subjects. (Fig23)

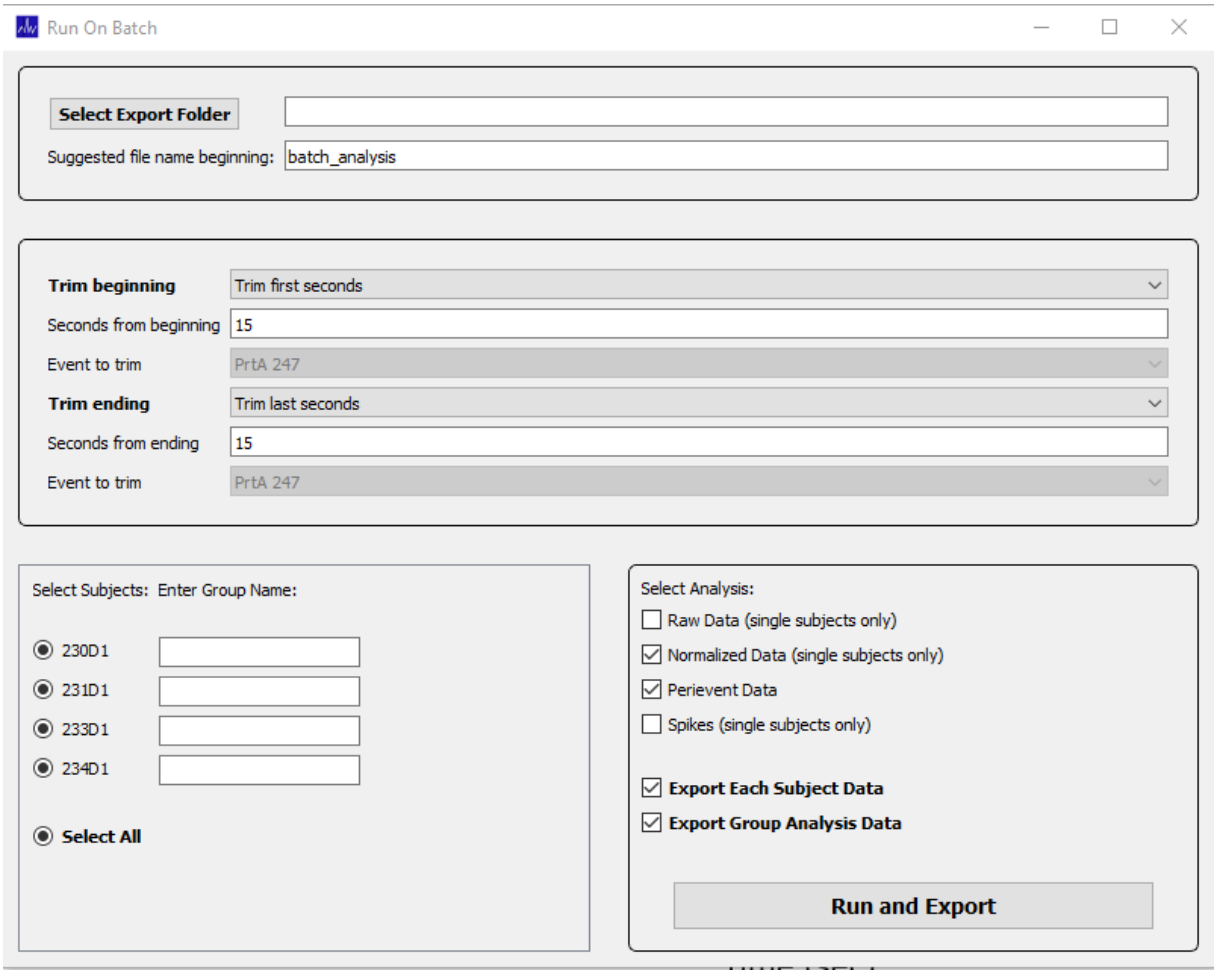


Fig23. Run on Batch window for multiple subjects data analysis.

The subjects can be selected within this window. However, the user also can “remember” interesting subjects by checking “Include Subject” checkbox at the bottom of each subject plot. Selected subjects will automatically appear as checked in the “Run on Batch” window.

Importantly, here the same settings for trimming, downsampling, and data normalization/smoothing will be applied to all selected subjects.

If the user selected any event in the Options next to the subject plots, the selected event will be shown on the plots.

“Export Each Subject Data” will save csv files with the selected single subject analyzed data as well as the plots.

“Export Group Analysis Data” will save csv files with group analysis data as well as the plots. (Fig24)

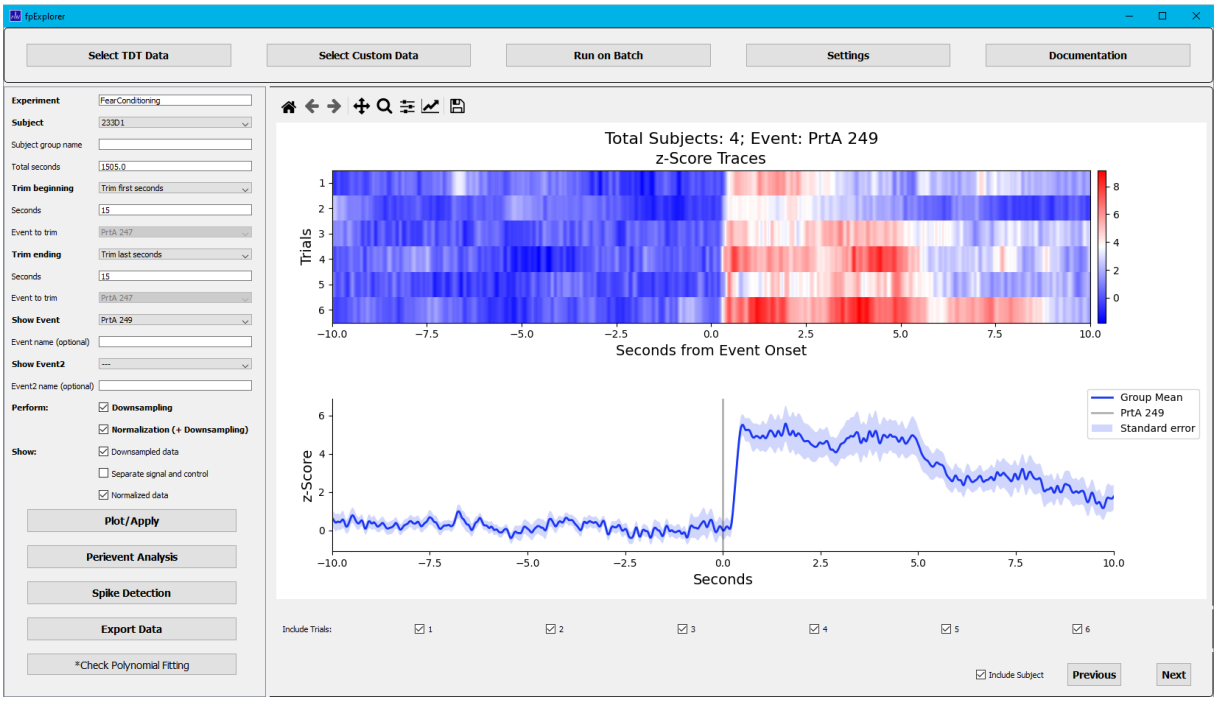


Fig24. Example of batch analysis showing peri-event results averaged across 4 subjects.

3. Analysis Methods

3.1. Normalization

Before each normalization, data is downsampled by default. Then, the user has two methods to choose from when it comes to polynomial fitting. The following methods allow to fit the control to the signal, in order to correct for the effects of photobleaching and movement artifacts.

The application relies on Python's numpy package to calculate polyfit.

(https://numpy.org/doc/stable/reference/generated/numpy.polynomial.polynomial.Polynomial.fit.html

https://numpy.org/doc/stable/reference/generated/numpy.polynomial.polynomial.polyval.html#numpy.polynomial.polynomial.polyval)

Before we perform the fitting, we clean the signal from the values that are above or below 2x standard deviations of the mean. That allows us to get a more reliable polynomial fit.

3.1.1. Standard Polynomial Fitting (linear regression of control channel onto signal channel)

This method assumes an equal decrease in signal in both channels over time, and importantly, there also shouldn't be an obvious dip in the isosbestic control signal whenever there is a large spike in signal (such dip can arise because the wavelength of the LED used for the control channel tends to be slightly to the left of the isosbestic point of fluorescent sensors).

The standard in the field has been to perform linear polynomial fitting of the isosbestic control onto the measured signal and use this fitted signal as "F0". Then calculate "deltaF" by subtracting F0 from the signal at each time point. This method has for instance been described in e.g. Lerner et al. 2015, Cell 162. We based our code for this normalization method on the analysis tool developed by Dr. David

Barker (Rutgers University) (pMAT, Photometry Modular Analysis Tool). Note that the following method is used after smoothing.*

Pseudo code:

```
bls = polyfit(control, GCaMP signal,1)

fit_line = multiply(bls[0], control) + bls[1]

dFF = (GCaMP signal - fit_line)/fit_line * 100
```

We decided to modify this method after some normalization problems when control channel started below signal and ended above signal. Numpy Python package was used for calculations and programming solution explained on the Stackoverflow forum.**

Pseudo code:

```
mu = mean(control)

std = std(control)

# call numpy.polynomial.polynomial.Polynomial.fit, using the shifted and scaled version of control

cscaled = Polynomial.fit ((control - mu)/std, signal, 1)

# create a poly1d object that can be called

pscaled = Polynomial(cscaled.convert().coef)

# inputs to pscaled must be shifted and scaled using mu and std

F0 = pscaled((control - mu)/std)

dffnorm = (signal - F0)/F0 * 100

# find all values of the normalized DF/F that are negative so you can next shift up the curve

# to make 0 the mean value for DF/F

negative = dffnorm[dffnorm<0]

dff = dffnorm - mean(negative)
```

**https://github.com/djamesbarker/pMAT*

***https://stackoverflow.com/questions/45338872/matlab-polyval-function-with-three-outputs-equivalent-in-python-numpy*

What polyval with the extra mu-input in Matlab does is to use the polynomial function obtained with polyfit using a transformed (z-score transformation) x-vector that is scaled (i.e. std=1) and centered (i.e. mean=0), apply this polynomial function to the transformed x-vector to regress it onto the y-vector, and then "untransform" the regressed x-vector so that its values are in a numerical domain again (like the original x and y vectors) instead of in the z-score domain. The code on the forum should do exactly that, it's just more lines of code because it's basically writing things out step by step instead of using an all-in-one function in Matlab.

3.1.2. Modified Polynomial Fitting (linear regression of data onto time axis)

Dr. Patrick Mulholland (MUSC) noticed that often fluorescent signals in both channels do not decrease over time at an equal rate. Therefore, he has suggested to correct both signals for decay over time by fitting them independently onto the time vector. Then he calculates deltaF for both channels, and then

he substracts deltaF of the control channel from deltaF of the signal channel to get a normalized signal. See e.g. Braunscheidel et al. 2019, J Neuroscience 39(46) for reference to this normalization method.

Pseudo code:

```
# fit time axis to the GCaMP stream (numpy.polynomial.polynomial.Polynomial.fit)

bls_Ca = Polynomial.fit (time, GCaMP signal,1)

# numpy.polynomial.polynomial.polyval

F0Ca = polyval(time,bls_Ca.convert().coef)

# dF/F for the GCaMP channel

dFFCa = (GCaMP signal - F0Ca)/F0Ca *100

# fit time axis to the control stream

bls_ref = Polynomial.fit (time, control,1)

F0Ref = polyval(time,bls_ref.convert().coef)

# dF/F for the control channel

dFFRef = (control - F0Ref)/F0Ref *100

dFFnorm = dFFCa - dFFRef

# find all values of the normalized DF/F that are negative so you can next shift up the curve

# to make 0 the mean value for DF/F

negative = dFFnorm[dFFnorm<0]

dFF = dFFnorm - mean(negative)
```

3.1.3. Custom baseline

In some cases when performing whole trace analysis, one might prefer to select one's own custom baseline period to perform the normalization, instead of using the entire trace. Doing this can for instance make sense when assessing the effects of a drug injection on the fiber photometry signal, where there is a clear drug-free baseline period followed by a post-injection period where the drug may have affected neural activity. The application allows users to select that custom baseline duration. (Fig.25)

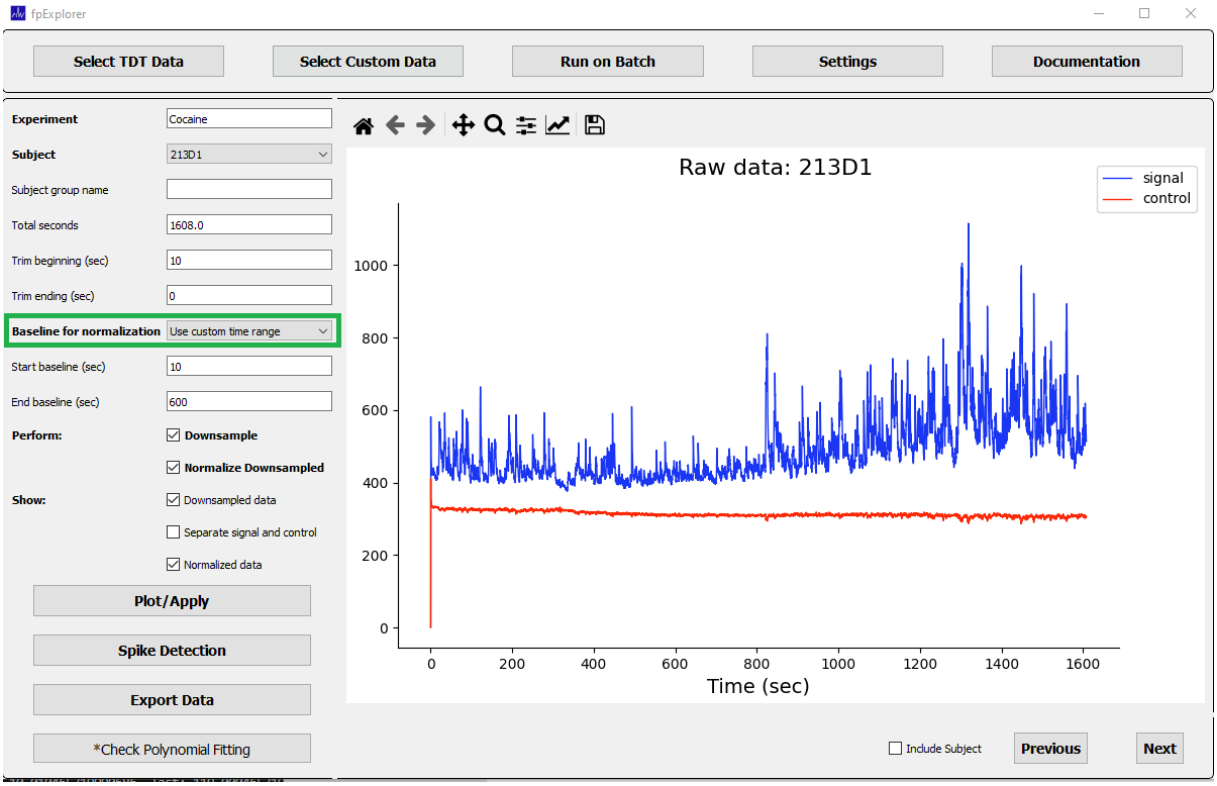


Fig25. Option to select custom baseline for the normalization.

To improve the polynomial fitting to the shorter selected baseline section, we slightly modified the polynomial fitting methods described above. Before we look for the best fit, we clean the signal from the values that are above or below 2x standard deviations of the mean of signal value during the baseline period.

Pseudo code for standard polynomial fitting:

```
# find mean and standard deviation of the

mean_baseline_signal = mean(signal_baseline_arr)

stdev_baseline_signal = std(signal_baseline_arr)

indexes = where((signal_baseline_arr < mean_baseline_signal + 2*stdev_baseline_signal) &
(signal_baseline_arr > mean_baseline_signal - 2*stdev_baseline_signal))

selected_signal = signal_baseline_arr[indexes]

selected_control = control_baseline_arr[indexes]

mu = mean(selected_control)

std = std(selected_control)

# call numpy.polynomial.polynomial.Polynomial.fit, using the shifted and scaled version of control

cscaled = Polynomial.fit ((selected_control - mu)/std, selected_signal, 1)

# create a poly1d object that can be called

pscaled = Polynomial(cscaled.convert().coef)

# inputs to pscaled must be shifted and scaled using mu and std

F0 = pscaled((control_arr - mu)/std)

dffnorm = (signal_arr - F0)/F0 * 100
```

Pseudo code for modified polynomial fitting:

```
mean_baseline_signal = mean(signal_baseline_arr)

indexes = where((signal_baseline_arr < mean_baseline_signal + 2*stdev_baseline_signal) &
(signal_baseline_arr > mean_baseline_signal - 2*stdev_baseline_signal))

selected_signal = signal_baseline_arr[indexes]

selected_control = control_baseline_arr[indexes]

# create a list of sample indexes for the time variable

selected_ts_baseline_arr = [0 till length(selected_signal)]

# fit time axis to the 465nm stream

bls_Ca = Polynomial.fit (selected_ts_baseline_arr,selected_signal,1)

F0Ca = polyval(ts_arr,bls_Ca.convert().coef)

# dF/F for the 465 channel

dFFCa = (signal_arr - F0Ca)/F0Ca *100

bls_ref = Polynomial.fit  (selected_ts_baseline_arr,selected_control,1)

F0Ref = polyval(ts_arr,bls_ref.convert().coef)

# dF/F for the 405 channel

dFFRef = (control_arr - F0Ref)/F0Ref *100

dFFnorm = dFFCa – dFFRef
```

While looking for the best normalization for the current trace (this is done either by changing “Standard Polynomial Fitting” in the settings to “Modified Polynomial Fitting” or the other way round, or selecting different custom baseline durations), we recommend to always check the fitting (Fig26).

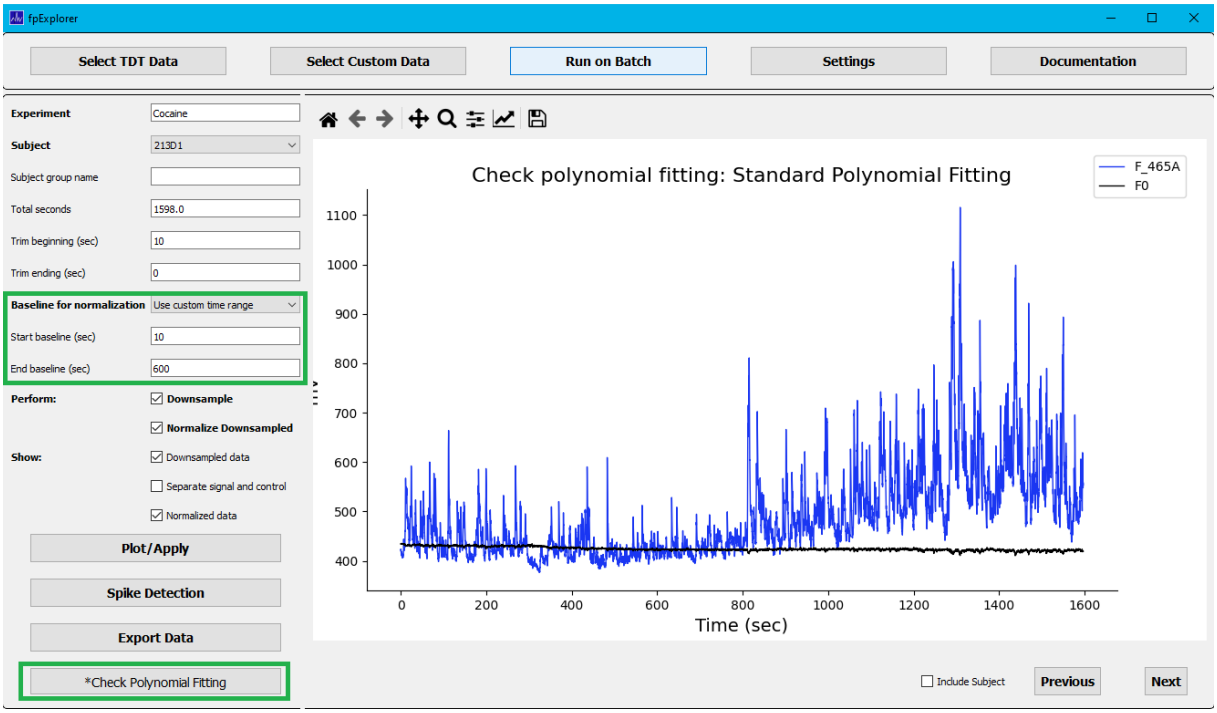


Fig26. One should always check the fitting to see which parameters are best for the trace.

3.2. Z-score

While optional for looking at whole trace data, Z-score transformation has to be applied within the perievent analysis to be able to better compare data from different trials with each other. It is performed on normalized data. Signals are converted into a robust z-score.

Pseudo code:

```
mad = median_abs_deviation(dFF[baseline])

zscore = ((dFF - median(dFF[baseline]))/mad)
```

3.2.1. Z-score group analysis

To calculate group Z-score, mean values from all trials from all subjects (not from means from each subject) are calculated as well as standard errors. There is also an option to plot each subject's mean values from that subject's trials instead of standard errors.

Heat map plot represents each trial's mean values from all analyzed subjects.

3.3. Area Under the Curve (AUC)

The application calculates area under the curve using Python's sklearn package. It represents the area under z-scored values.
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.auc.html

3.3.1. Area Under the Curve (AUC) group analysis

For group analysis, area under the mean value from all subject's z-scores is calculated.
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.auc.html

3.4. Spike detection

The algorithms used in the application to detect spikes are explained here:

https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html

scipy.signal.find_peaks(x, height=None, threshold=None, distance=None, prominence=None, width=None, wlen=None, rel_height=0.5, plateau_size=None)

Find peaks inside a signal based on peak properties. This function takes a 1-D array and finds all local maxima by simple comparison of neighboring values. Optionally, a subset of these peaks can be selected by specifying conditions for a peak's properties.

Parameters

x: sequence

A signal with peaks.

height: number or ndarray or sequence, optional

Required height of peaks. Either a number, None, an array matching x or a 2-element sequence of the former. The first element is always interpreted as the minimal and the second, if supplied, as the maximal required height.

threshold: number or ndarray or sequence, optional

Required threshold of peaks, the vertical distance to its neighboring samples. Either a number, None, an array matching x or a 2-element sequence of the former. The first element is always interpreted as the minimal and the second, if supplied, as the maximal required threshold.

distance: number, optional

Required minimal horizontal distance (>= 1) in samples between neighbouring peaks. Smaller peaks are removed first until the condition is fulfilled for all remaining peaks.

prominence: number or ndarray or sequence, optional

Required prominence of peaks. Either a number, None, an array matching x or a 2-element sequence of the former. The first element is always interpreted as the minimal and the second, if supplied, as the maximal required prominence.

width: number or ndarray or sequence, optional

Required width of peaks in samples. Either a number, None, an array matching x or a 2-element sequence of the former. The first element is always interpreted as the minimal and the second, if supplied, as the maximal required width.

wlen: int, optional

Used for calculation of the peaks prominences, thus it is only used if one of the arguments prominence or width is given. See argument wlen in peak_prominences for a full description of its effects.

rel_height: float, optional

Used for calculation of the peaks width, thus it is only used if width is given. See argument rel_height in peak_widths for a full description of its effects.

plateau_size: number or ndarray or sequence, optional

Required size of the flat top of peaks in samples. Either a number, None, an array matching x or a 2-element sequence of the former. The first element is always interpreted as the minimal and the second, if supplied as the maximal required plateau size.