neurotic Documentation

Release 1.4.0

Jeffrey Gill

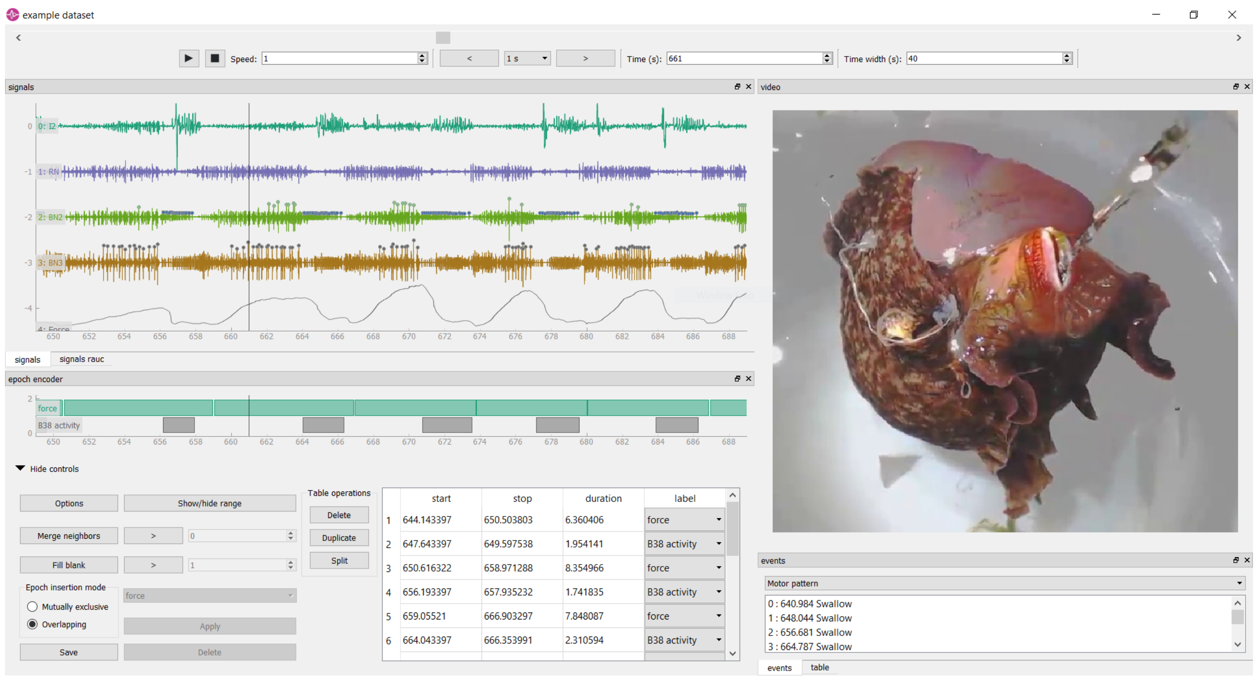
Mar 04, 2020

Table of Contents

Curate, visualize, annotate, and share your behavioral ephys data using Python

Version: 1.4.0 (other versions)

neurotic is an app for Windows, macOS, and Linux that allows you to easily review and annotate your electrophysiology data and simultaneously captured video. It is an easy way to load your Neo-compatible data (see `neo.io` for file formats) into ephyviewer without doing any programming. Share a single metadata file with your colleagues and they too will quickly be looking at the same datasets!



	start	stop	duration	label
1	644.143397	650.503803	6.360406	force
2	647.643397	649.597538	1.954141	B38 activity
3	650.616322	658.971288	8.354966	force
4	656.193397	657.935232	1.741835	B38 activity
5	659.05521	666.903297	7.848087	force
6	664.043397	666.353991	2.310594	B38 activity

0 : 640.984 Swallow
1 : 648.044 Swallow
2 : 656.681 Swallow
3 : 664.787 Swallow

CHAPTER 1

Overview

To use **neurotic**, first organize your datasets in a *metadata file* like this (see *Configuring Metadata*):

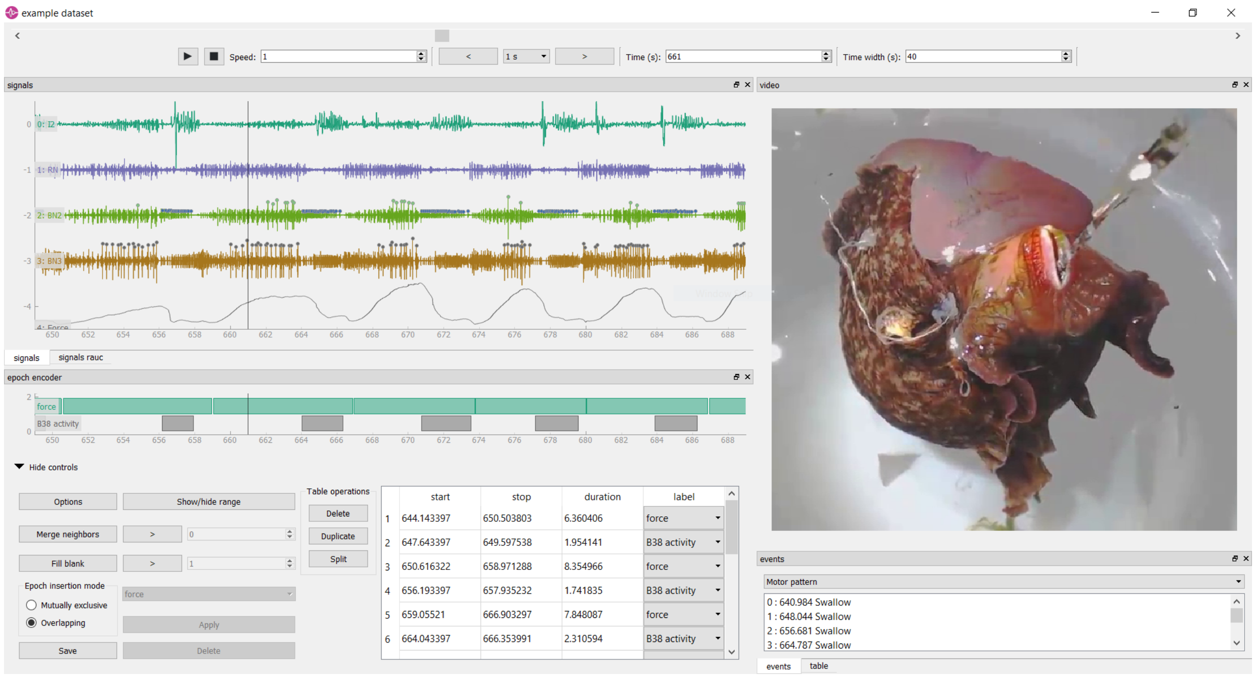
```
my favorite dataset:
    description: This time it actually worked!

    data_dir:           C:\local_dir_containing_files
    remote_data_dir:    http://myserver/remote_dir_containing_downloadable_files  #␣
↪optional
    data_file:          data.axgx
    video_file:         video.mp4
    # etc

    video_offset: -3.4  # seconds between start of video and data acq
    epoch_encoder_possible_labels:
        - label01
        - label02
    plots:
        - channel: I2
          ylim: [-30, 30]
        - channel: RN
          ylim: [-60, 60]
        # etc

    filters:  # used only if fast loading is off (lazy=False)
        - channel: Force
          lowpass: 50
        # etc
    amplitude_discriminators:  # used only if fast loading is off (lazy=False)
        - name: B3 neuron
          channel: BN2
          units: uV
          amplitude: [50, 100]
        # etc
```

(continues on next page)

(continued from previous page)

```
another dataset:
    # etc
```

Open your metadata file in **neurotic** and choose a dataset. If the data and video files aren't already on your local computer, the app can download them for you, even from a password-protected server. Finally, click launch and the app will use a standard viewer layout to display your data to you using ephyviewer.



	start	stop	duration	label
1	644.143397	650.503803	6.360406	force
2	647.643397	649.597538	1.954141	B38 activity
3	650.616322	658.971288	8.354966	force
4	656.193397	657.935232	1.741835	B38 activity
5	659.05521	666.903297	7.848087	force
6	664.043397	666.353991	2.310594	B38 activity

0 : 640.984 Swallow
1 : 648.044 Swallow
2 : 656.681 Swallow
3 : 664.787 Swallow

(Pictured above is a voracious Aplysia californica in the act of making the researcher very happy.)

The viewers are easy and intuitive to navigate (see User Interface):

- Pressing the play button will scroll through your data and video in real time, or at a higher or lower rate if the speed parameter is changed.
- The arrow/WASD keys allow you to step through time in variable increments.
- Jump to a time by clicking on an event in the event list or a table entry in the epoch encoder.
- To show more or less time at once, right-click and drag right or left to contract or expand time.
- Scroll the mouse wheel in the trace viewer or video viewer to zoom.
- The epoch encoder can be used to block out periods of time during which something interesting is happening for later review or further analysis (saved to a CSV file).
- All panels can be hidden, undocked, stacked, or repositioned on the fly.

Electrophysiologists will find this tool useful even if they don't need the video synchronization feature!

Portability is easy with neurotic! Use relative paths in your metadata file along with a remotely accessible data store such as GIN to make your metadata file fully portable. The same metadata file can be copied to a different computer, and downloaded files will automatically be saved to the right place. Data stores can be password protected and **neurotic** will prompt you for a user name and password. This makes it easy to share the **neurotic** experience with your colleagues!

CHAPTER 2

Installing neurotic

neurotic requires Python 3.6 or later.

Note that the latest release of one of **neurotic**'s dependencies, pyqtgraph 0.10.0, is incompatible with Python 3.8 or later on Windows unless that dependency is installed via conda-forge (recommended method) (details).

2.1 Standalone Installers (recommended for beginners)

Downloadable installers make installing **neurotic** easy for beginners. When available, they can be downloaded from the GitHub Releases page:

> Download installers here (listed under "Assets")

These installers are intended for users who do not want to independently install Python or conda just to use **neurotic**. They will install **neurotic** and everything it needs (including a fully contained Python environment) into a dedicated directory on your computer. On Windows, the installer will also create a Start Menu shortcut for launching the app.

Because the process of building installers is not automated, installers may not be available for the latest releases for all platforms.

For developers, a recipe for building new installers using conda constructor is maintained here: constructor recipe.

2.2 Alternate Method: conda (recommended for Pythonistas)

conda users can install **neurotic** and all of its dependencies with one command:

```
conda install -c conda-forge neurotic
```

On Windows, this will also create a Start Menu shortcut for launching the app.

2.3 Alternate Method: pip

Install **neurotic** from PyPI using

```
pip install neurotic
```

Note that installation via `pip` skips one dependency: PyAV, which is required for displaying videos, and without which **neurotic** will ignore videos. PyAV is not easily installed with `pip` on some systems, especially Windows. The easiest way to separately install PyAV is using conda:

```
conda install -c conda-forge av
```

CHAPTER 3

Updating neurotic

The recommended method of updating **neurotic** depends on the original method of installation.

If you are unsure what method you used, updating using `conda` or `pip` is likely to work. Standalone installers may be safe too, though this could lead to having multiple version installed simultaneously.

3.1 Updating with Standalone Installers

If you previously installed **neurotic** using a standalone installer, you may install a newer version using another installer, either into a different directory or by first uninstalling the old version. Installers can be downloaded from the GitHub Releases page:

> Download installers here (listed under "Assets")

Alternatively, if a new installer is not currently available for your platform, or if you would just like a much faster method, you may use the command line tools provided by the installer (via the "Anaconda Prompt" on Windows, or the Terminal on macOS and Linux):

```
conda update -c conda-forge neurotic
```

3.2 Updating with conda

If you installed **neurotic** with conda, you can update to the latest release using

```
conda update -c conda-forge neurotic
```

3.3 Updating with pip

If you installed **neurotic** using `pip`, you can update to the latest release available on PyPI using

```
pip install -U neurotic
```

3.4 Development Version

If you are interested in trying new, unreleased features of **neurotic**, you may install the latest development version from GitHub using

```
pip install -U git+https://github.com/jpgill86/neurotic.git
```

Note that if you install the development version, you may also need the latest development version of ephyviewer, which you can get using

```
pip install -U git+https://github.com/NeuralEnsemble/ephyviewer.git
```

CHAPTER 4

Getting Started

If you installed **neurotic** into a conda environment, first activate it:

```
conda activate <environment name>
```

Launch the app from the command line:

```
neurotic
```

A simple example is provided. Select the "example dataset", download the associated data (~7 MB), and then click "Launch". See User Interface for help with navigation.

Disabling "Fast loading" before launch will enable additional features including amplitude-threshold spike detection and signal filtering.

To inspect the metadata file associated with the example or to make changes to it, click "Edit metadata". See *Configuring Metadata* for details about the format.

If you prefer Jupyter notebooks, you can launch an example notebook instead, which includes a tutorial for using **neurotic**'s API:

```
neurotic --launch-example-notebook
```

The command line interface accepts other arguments too:

```
usage: neurotic [-h] [-V] [--debug] [--no-lazy] [--thick-traces]
                [--show-datetime] [--ui-scale {tiny,small,medium,large,huge}]
                [--theme {light,dark,original,printer-friendly}]
                [--launch-example-notebook]
                [file] [dataset]

neurotic lets you curate, visualize, annotate, and share your behavioral ephys
data.

positional arguments:
  file                  the path to a metadata YAML file (default: an example
```

(continues on next page)

(continued from previous page)

```
                        file)
  dataset               the name of a dataset in the metadata file to select
                        initially (default: the first entry in the metadata
                        file)

optional arguments:
  -h, --help            show this help message and exit
  -V, --version         show program's version number and exit
  --debug               enable detailed log messages for debugging
  --no-lazy             do not use fast loading (default: use fast loading)
  --thick-traces        enable support for traces with thick lines, which has
                        a performance cost (default: disable thick line
                        support)
  --show-datetime       display the real-world date and time, which may be
                        inaccurate depending on file type and acquisition
                        software (default: do not display)
  --ui-scale {tiny,small,medium,large,huge}
                        the scale of user interface elements, such as text
                        (default: medium)
  --theme {light,dark,original,printer-friendly}
                        a color theme for the GUI (default: light)
  --launch-example-notebook
                        launch Jupyter with an example notebook instead of
                        starting the standalone app (other args will be
                        ignored)
```

CHAPTER 5

Configuring Metadata

To load your data with **neurotic**, you must organize them in one or more YAML files, called *metadata files*.

YAML files are very sensitive to punctuation and indentation, so mind those details carefully! Importantly, the tab character cannot be used for indentation; use spaces instead. There are many free websites that can validate YAML for you.

You may include comments in your metadata file, which should begin with `#`.

5.1 Top-Level Organization

Datasets listed within the same metadata file must be given unique names, which may include spaces. The special name `neurotic_config` is reserved for **neurotic** configuration settings and cannot be used for datasets.

In addition to names, a long description can be provided for each dataset.

Details pertaining to each dataset, including the description, are nested beneath the dataset name using indentation. You may need to use double quotes around names, descriptions, or other text if they contain special characters (such as `:` or `#`) or are composed only of numbers (such as a date).

```
experiment 2020-01-01:
    description: Both the name and description will be visible when neurotic loads␣
↪the metadata
    # other details about this dataset will go here

my favorite dataset:
    description: This time it actually worked!
    # other details about this dataset will go here
```

5.2 Specifying Data Locations

Within a dataset's YAML block, paths to data and video files should be provided.

All files associated with a dataset should be collected into a single directory. A path to the local copy of this directory can be provided using the `data_dir` key. You may specify `data_dir` as an absolute path (e.g., `C:\Users\me\folder`) or as a path relative to the metadata file (e.g., `folder`). If left unspecified, the directory containing the metadata file is used.

Paths to individual files within the dataset are provided using keys listed below. These paths should be given relative to `data_dir`. If `data_dir` is flat (no subdirectories), these should be simply the file names.

Key	Description
`data_file`	A single Neo-compatible data file (see `neo.io` for file formats)
`video_file`	A video file that can be synchronized with `data_file`
`annotations_file`	A CSV file for read-only annotations
`epoch_encoder_file`	A CSV file for annotations writable by the epoch encoder
`tridesclous_file`	A CSV file output by tridesclous's `DataIO.export_spikes`

Note that the `annotations_file` must contain exactly 4 columns with these headers: "Start (s)", "End (s)", "Type", and "Label".

The `epoch_encoder_file` must contain exactly 3 columns with these headers: "Start (s)", "End (s)", and "Type". (The fourth column is missing because ephyviewer's epoch encoder is currently unable to attach notes to individual epochs; this may be improved upon in the future.)

The `tridesclous_file` is described in more detail in *tridesclous Spike Sorting Results*.

5.3 Remote Data Available for Download

Data files must be stored on the local computer for **neurotic** to load them and display their contents. If the files are available for download from a remote server, **neurotic** can be configured to download them for you to the local directory specified by `data_dir` if the files aren't there already.

Specify the URL to the directory containing the data on the remote server using `remote_data_dir`. **neurotic** expects the local `data_dir` and the `remote_data_dir` to have the same structure and will mirror the `remote_data_dir` in the local `data_dir` when you download data (not a complete mirror, just the specified files).

For an example, consider the following:

```
my favorite dataset:
    data_dir:           C:\Users\me\folder
    remote_data_dir:    http://myserver/remote_folder
    data_file:          data.axgx
    video_file:         video.mp4
```

With a metadata file like this, the file paths `data_file` and `video_file` are appended to `remote_data_dir` to obtain the complete URLs for downloading these files, and they will be saved to the local `data_dir`.

If you have many datasets hosted by the same server, you can specify the server URL just once using the special `remote_data_root` key, which should be nested under the reserved name `neurotic_config` outside of any dataset's YAML block. This allows you to provide for each dataset a partial URL to a folder in `remote_data_dir` which is relative to `remote_data_root`. For example:

```
neurotic_config:  # reserved name for global settings
    remote_data_root:   http://myserver

my favorite dataset:
```

(continues on next page)

(continued from previous page)

```
    data_dir:            C:\Users\me\folder1
    remote_data_dir:     remote_folder1
    data_file:           data.axgx
    video_file:          video.mp4

another dataset:
    data_dir:            C:\Users\me\folder2
    remote_data_dir:     remote_folder2
    data_file:           data.axgx
    video_file:          video.mp4
```

Here, URLs to video files are composed by joining `remote_data_root` + `remote_data_dir` + `video_file`.

Recall that if `data_dir` is a relative path, it is assumed to be relative to the metadata file. In the example above, if the metadata file is located in `C:\Users\me`, the paths could be abbreviated:

```
neurotic_config:
    remote_data_root:   http://myserver

my favorite dataset:
    data_dir:            folder1
    remote_data_dir:     remote_folder1
    data_file:           data.axgx
    video_file:          video.mp4

another dataset:
    data_dir:            folder2
    remote_data_dir:     remote_folder2
    data_file:           data.axgx
    video_file:          video.mp4
```

Note: Portability is easy with neurotic! Use relative paths in your metadata file along with a remotely accessible data store such as GIN to make your metadata file fully portable. The example above is a simple model of this style. A metadata file like this can be copied to a different computer, and downloaded files will automatically be saved to the right place. Data stores can be password protected and **neurotic** will prompt you for a user name and password. This makes it easy to share the **neurotic** experience with your colleagues!

5.3.1 URLs to Use with GIN

If you have data stored in a **public** repository on GIN, you can access it from a URL of this form:

```
https://gin.g-node.org/<username>/<reponame>/raw/master/<path>
```

For **private** repositories, you must use a different URL that takes advantage of the WebDAV protocol:

```
https://gin.g-node.org/<username>/<reponame>/_dav/<path>
```

The second form works with public repos too, but GIN login credentials are still required. Consequently, the first form is more convenient for public repos.

5.4 Global Configuration Settings

The top-level name `neurotic_config` is reserved for configuration settings that apply to all datasets or to the app itself. Presently, only one configuration setting is implemented, but future versions of **neurotic** may add more under this name.

Key	Description
`remote_data_root`	A URL prepended to each `remote_data_dir` that is not already a full URL (i.e., does not already begin with a protocol scheme like `https://`)

For example:

```
neurotic_config:
    remote_data_root:   http://myserver

my favorite dataset:
    # dataset details here
```

5.5 Data Reader (Neo) Settings

The electrophysiology file specified by `data_file` is read using Neo, which supports many file types. A complete list of the implemented formats can be found here: `neo.io`.

By default, **neurotic** will use the file extension of `data_file` to guess the file format and choose the appropriate Neo IO class for reading it. If the guess fails, you can force **neurotic** to use a different class by specifying the class name with the `io_class` parameter (all available classes are listed here: `neo.io`).

Some Neo IO classes accept additional arguments beyond just a filename (see the Neo docs for details: `neo.io`). You can specify these arguments in your metadata using the `io_args` parameter.

For example, suppose you have data stored in a plain text file that is missing a file extension. The `neo.io.AsciiSignalIO` class can read plain text files, but you must specify this manually using `io_class` because the extension is missing. You could do this and pass in supported arguments in the following way:

```
my favorite dataset:
    data_file: plain_text_file_without_file_extension

    io_class: AsciiSignalIO

    io_args:
        skiprows: 1 # skip header
        delimiter: ' ' # space-delimited
        t_start: 5 # sec
        sampling_rate: 1000 # Hz
        units: mV
```

5.6 Video Synchronization Parameters

5.6.1 Constant Offset

If data acquisition began with some delay after video capture began, provide a negative value for `video_offset` equal to the delay in seconds. If video capture began after the start of data acquisition, use a positive value. A value of zero will have no effect.

neurotic warns users about the risk of async if `video_file` is given but `video_offset` is not. To eliminate this warning for videos that have no delay, provide zero.

5.6.2 Frame Rate Correction

If the average frame rate reported by the video file is a little fast or slow, you may notice your video and data going out of sync late in a long experiment. You can provide the `video_rate_correction` parameter to fix this. The reported average frame rate of the video file will be multiplied by this factor to obtain a new frame rate used for playback. A value less than 1 will decrease the frame rate and shift video events to later times. A value greater than 1 will increase the frame rate and shift video events to earlier times. A value of 1 has no effect.

You can obtain a good estimate of what value to use by taking the amount of time between two events in the video and dividing by the amount of time between the same two events according to the data record (seen, for example, as synchronization pulses or as movement artifacts).

5.6.3 Discrete Desynchronization Events

If you paused data acquisition during your experiment while video capture was continuous, you can use the `video_jumps` parameter to correct for these discrete desynchronization events, assuming you have some means of reconstructing the timing. For each pause, provide an ordered pair of numbers in seconds: The first is the time *according to data acquisition* (not according to the video) when the pause occurred, and the second is the duration of the pause during which the video kept rolling.

For example:

```
my favorite dataset:
    video_file: video.mp4
    # etc

    video_jumps:
        # a list of ordered pairs containing:
        # (1) time in seconds when paused occurred according to DAQ
        # (2) duration of pause in seconds
        - [60, 10]
        - [120, 10]
        - [240, 10]
```

These values could correct for three 10-second pauses occurring at times 1:00, 2:00, 3:00 according to the DAQ, which would correspond to times 1:00, 2:10, 3:20 according to the video. The extra video frames captured during the pauses will be excised from playback so that the data and video remain synced.

neurotic will automatically suggest values for `video_jumps` if it reads an AxoGraph file that contains stops and restarts (only if `video_jumps` is not already specified).

5.7 Real-World Date and Time

The GUI can optionally display the real-world date and time. This feature is accurate only if the recording is continuous (no interruptions or pauses during recording) and the start time of the recording is known. Some data file formats may store the start time of the recording, in which case **neurotic** will use that information automatically. However, if the start time is missing or inaccurate, it can be specified in the metadata like this:

```
my favorite dataset:
    data_file: data.axgx
    rec_datetime: 2020-01-01 13:14:15
    # etc
```

5.8 Plot Parameters

Use the `plots` parameter to specify which signal channels from `data_file` you want plotted and how to scale them. Optionally, a color may be specified for channels using a single letter color code (e.g., `'b'` for blue or `'k'` for black) or a hexadecimal color code (e.g., `'1b9e77'`).

Consider the following example, and notice the use of hyphens and indentation for each channel.

```
my favorite dataset:
    data_file: data.axgx
    # etc

    plots:
        - channel: Extracellular
          ylabel: Buccal nerve 2 (BN2)
          units: uV
          ylim: [-150, 150]
          color: r

        - channel: Intracellular
          ylabel: B3 neuron
          units: mV
          ylim: [-100, 50]
          color: '666666'

        - channel: Force
          units: mN
          ylim: [-10, 500]
```

This would plot the "Extracellular", "Intracellular", and "Force" channels from the `data_file` in the given order. `ylabel` is used to relabel a channel and is optional. The `units` and `ylim` parameters are used together to scale each signal such that the given range fits neatly between the traces above and below it. If `units` is not given, they are assumed to be microvolts for voltage signals and millinewtons for force signals. If `ylim` is not given, they default to `[-120, 120]` for voltages and `[-10, 300]` for forces.

If `plots` is not provided, all channels are plotted using the default ranges, except for channels that match these patterns: "Analog Input #*" and "Clock". Channels with these names can be plotted if given explicitly by `plots`.

5.9 Time Range

The amount of time initially visible can be specified in seconds with `t_width`.

The position of the vertical line, which represents the current time in each plot, can be specified as a fraction of the plot range with `past_fraction`. A value of 0 places the vertical line at the left edge of each plot; consequently, everything plotted is "in the future", occurring after the current time. A value of 1 places the vertical line at the right edge of each plot; consequently, everything plotted is "in the past", coming before the current time. The default value of 0.3 causes the first 30% of the plot range to display "the past" and the last 70% to display "the future".

5.10 Epoch Encoder Parameters

The labels available to the epoch encoder must be specified ahead of time using `epoch_encoder_possible_labels` (this is a current limitation of ephyviewer that may eventually be improved upon).

For example:

```
my favorite dataset:
    epoch_encoder_file: epoch-encoder.csv
    # etc

    epoch_encoder_possible_labels:
        - label1
        - label2
        - label3
```

5.11 Filters

Highpass, lowpass, and bandpass filtering can be applied to signals using the `filters` parameter. Note that filters are only applied if fast loading is off (`lazy=False`).

Consider the following example, and notice the use of hyphens and indentation for each filter.

```
my favorite dataset:
    data_file: data.axgx
    # etc

    filters:  # used only if fast loading is off (lazy=False)

        - channel: Extracellular
          highpass: 300 # Hz
          lowpass: 500 # Hz

        - channel: Intracellular
          highpass: 300 # Hz

        - channel: Force
          lowpass: 50 # Hz
```

Filter cutoffs are given in hertz. Combining `highpass` and `lowpass` provides bandpass filtering.

5.12 Amplitude Discriminators

Spikes with peaks that fall within amplitude windows given by `amplitude_discriminators` can be automatically detected by **neurotic** on the basis of amplitude alone. Note that amplitude discriminators are only applied if fast loading is off (`lazy=False`).

Detected spikes are indicated on the signals with markers, and spike trains are displayed in a raster plot. Optionally, a color may be specified for an amplitude discriminator using a single letter color code (e.g., `'b'` for blue or `'k'` for black) or a hexadecimal color code (e.g., `'1b9e77'`).

In addition to restricting spike detection for a given unit to an amplitude window, detection can also be limited in time to overlap with epochs with a given label.

Consider the following example, and notice the use of hyphens and indentation for each amplitude discriminator.

```
my favorite dataset:
    data_file: data.axgx
    # etc

    amplitude_discriminators:  # used only if fast loading is off (lazy=False)

        - name: Unit 1
          channel: Extracellular
          units: uV
          amplitude: [50, 150]
          color: r

        - name: Unit 2
          channel: Extracellular
          units: uV
          amplitude: [20, 50]
          epoch: Unit 2 activity
          color: 'e6ab02'
```

Here two units are detected on the same channel with different amplitude windows. Any peaks between 50 and 150 microvolts on the "Extracellular" channel will be tagged as a spike belonging to "Unit 1". The discriminator for "Unit 2" provides the optional `epoch` parameter. This restricts detection of "Unit 2" to spikes within the amplitude window that occur at the same time as epochs labeled "Unit 2 activity". These epochs can be created by the epoch encoder (reload required to rerun spike detection at launch-time), specified in the read-only `annotations_file`, or even be contained in the `data_file` if the format supports epochs.

Amplitude windows are permitted to be negative.

5.13 tridesclous Spike Sorting Results

tridesclous is a sophisticated spike sorting toolkit. The results of a sorting process can be exported to a CSV file using tridesclous's `DataIO.export_spikes` function. This file contains two columns: the first is the sample index of a spike, and the second is the ID for a cluster of spikes. If this file is specified with `tridesclous_file`, then a mapping from the cluster IDs to channels must be provided with `tridesclous_channels`.

In the following example, notice the lack of hyphens:

```
my favorite dataset:
    data_file: data.axgx
    tridesclous_file: spikes.csv
    # etc

    tridesclous_channels:
        0: [Channel A, Channel B]
        1: [Channel A]
        2: [Channel B]
        3: [Channel B]
        # etc
```

Here numeric cluster IDs are paired with a list of channels found in `data_file` on which the spikes were detected.

To show only a subset of clusters or to merge clusters, add the `tridesclous_merge` parameter.

In this example, note again the punctuation:

```
my favorite dataset:
    data_file: data.axgx
    tridesclous_file: spikes.csv
    # etc

    tridesclous_channels:
        0: [Channel A, Channel B]
        1: [Channel A]
        2: [Channel B]
        3: [Channel B]
        # etc

    tridesclous_merge:
        - [0, 1]
        - [3]
```

Now clusters 0 and 1 are combined into a single unit, and only that unit and cluster 3 are plotted; cluster 2 has been discarded.

5.14 Firing Rates

If spike trains were generated using *Amplitude Discriminators*, imported from *tridesclous Spike Sorting Results*, or included in the `data_file`, their smoothed firing rates can be computed. Note that firing rates are computed only if fast loading is off (`lazy=False`).

Firing rates are plotted as continuous signals. Colors are inherited from `amplitude_discriminators`, if they are provided there.

Firing rates are computed using a kernel that is convolved with the spike train. The metadata is specified like this:

```
my favorite dataset:
    data_file: data.axgx
    # etc

    amplitude_discriminators:  # used only if fast loading is off (lazy=False)

        - name: Unit 1
          channel: Extracellular
          units: uV
          amplitude: [50, 150]

    firing_rates:  # used only if fast loading is off (lazy=False)

        - name: Unit 1
          kernel: GaussianKernel
          sigma: 1.5 # sec
```

The elephant package's `instantaneous_rate` function is used for calculating firing rates. See `elephant.kernels` for the names of kernel classes that may be used with the `kernel` parameter. **neurotic** provides an additional kernel, *`CausalAlphaKernel`*, which may also be used. The `sigma` parameter is passed as an argument to the kernel class and should be given in seconds.

The rate calculation function and kernel classes are sourced from *`neurotic._elephant_tools`*, rather than the elephant package itself, to avoid requiring elephant as a package dependency.

5.15 Firing Frequency Burst Detectors

If spike trains were generated using *Amplitude Discriminators*, imported from *tridesclous Spike Sorting Results*, or included in the `data_file`, a simple burst detection algorithm that relies on instantaneous firing rate thresholds can be run to detect periods of intense activity. Note that burst detectors are only applied if fast loading is off (`lazy=False`).

Detected bursts are plotted as epochs. Colors are inherited from `amplitude_discriminators`, if they are provided there.

Burst detectors are specified in metadata like this:

```
my favorite dataset:
    data_file: data.axgx
    # etc

    amplitude_discriminators:  # used only if fast loading is off (lazy=False)

        - name: Unit 1
          channel: Extracellular
          units: uV
          amplitude: [50, 150]

    burst_detectors:  # used only if fast loading is off (lazy=False)

        - spiketrain: Unit 1
          name: Unit 1 burst  # optional, used for customizing output epoch name
          thresholds: [10, 8] # Hz
```

The algorithm works by scanning through the spike train with a name matching `spiketrain` (in this example, the spike train generated by the "Unit 1" amplitude discriminator). When the instantaneous firing frequency (IFF; note this is *NOT* the same as the *smoothed firing rate*, but rather the inverse of the inter-spike interval) exceeds the first threshold given (e.g., 10 Hz), a burst of activity is determined to start. After this, at the first moment when the IFF drops below the second threshold (e.g., 8 Hz), the burst is determined to end. After scanning through the entire spike train, many bursts that meet these criteria may be identified.

Note that in general the end threshold should not exceed the start threshold; this would essentially be the same as setting the start and end thresholds both to the greater value.

5.16 Rectified Area Under the Curve (RAUC)

One way to simplify a high-frequency signal is by plotted a time series of the rectified area under the curve (RAUC). Note that RAUCs are calculated only if fast loading is off (`lazy=False`).

For each signal, the baseline (mean or median) is optionally subtracted off. The signal is then rectified (absolute value) and divided into non-overlapping bins of fixed duration. Finally, the integral is calculated within each bin. The result is a new time series that represents the overall activity of the original signal. RAUC time series are plotted separately from the original signals in a second tab. Colors are inherited from `plots`, if they are provided there.

The choice of baseline is controlled by the `rauc_baseline` metadata parameter, which may have the value `None` (default), `'mean'`, or `'median'`. The size of the bins determines how smooth the RAUC time series is and is set by `rauc_bin_duration`, given in seconds. If `rauc_bin_duration` is not specified (default `None`), RAUC time series will not be calculated.

5.17 A Complete Example

These are the contents of the example metadata file that ships with **neurotic**, which can be loaded by running `neurotic` from the command line without arguments:

```
example dataset:
    description: This is an example data set

    # these data are a subset of Jeffrey Gill's dataset 2018-06-21_IN-VIVO_JG-08 002
    data_dir:           example-data
    remote_data_dir:    https://gin.g-node.org/jpgill86/neurotic-data/raw/master/
↪examples/example-data
    data_file:          data.axgx
    video_file:         video.mp4
    annotations_file:   annotations.csv
    epoch_encoder_file: epoch-encoder.csv

    video_offset: 640.3 # seconds

    epoch_encoder_possible_labels:
        - force
        - B38 activity

    plots:
        - channel: I2
          units: uV
          ylim: [-30, 30]

        - channel: RN
          units: uV
          ylim: [-60, 60]

        - channel: BN2
          units: uV
          ylim: [-120, 120]

        - channel: BN3
          units: uV
          ylim: [-150, 150]

        - channel: Force
          units: mN
          ylim: [-10, 300]

    filters:  # used only if fast loading is off (lazy=False)

        - channel: I2
          lowpass: 100 # Hz

        - channel: Force
          lowpass: 50 # Hz

    amplitude_discriminators:  # used only if fast loading is off (lazy=False)

        - name: B3
          channel: BN2
          units: uV
```

(continues on next page)

(continued from previous page)

```
            amplitude: [50, 150]
            color: '1b9e77'

          - name: B38
            channel: BN2
            units: uV
            amplitude: [17, 26]
            epoch: B38 activity
            color: '7570b3'

          - name: B4/B5
            channel: BN3
            units: uV
            amplitude: [85, 200]
            color: 'e6ab02'

    firing_rates:  # used only if fast loading is off (lazy=False)

          - name: B3
            kernel: CausalAlphaKernel
            sigma: 1.0

          - name: B38
            kernel: CausalAlphaKernel
            sigma: 1.0

          - name: B4/B5
            kernel: CausalAlphaKernel
            sigma: 1.0

    burst_detectors:  # used only if fast loading is off (lazy=False)

          - spiketrain: B3
            thresholds: [8, 2] # Hz

          - spiketrain: B38
            thresholds: [8, 5] # Hz

          - spiketrain: B4/B5
            thresholds: [3, 3] # Hz

    rauc_bin_duration: 0.1 # seconds, used only if fast loading is off (lazy=False)
```

CHAPTER 6

API Reference Guide

In addition to using **neurotic** as a standalone app, you can also leverage its API in your own code.

Note: **TL;DR**: The easiest way to use **neurotic** in an interactive session or script is by invoking *neurotic.quick_launch()*. For example:

```
>>> metadata = {'data_file': 'data.axgx'}
>>> neurotic.quick_launch(metadata=metadata)
```

or

```
>>> neurotic.quick_launch(blk=my_neo_block)
```

The core of the API consists of two classes and one function:

- *neurotic.datasets.metadata.MetadataSelector*: Read metadata files, download datasets
- *neurotic.datasets.data.load_dataset()*: Read datasets, apply filters and spike detection
- *neurotic.gui.config.EphyviewerConfigurator*: Launch ephyviewer

All public package contents are automatically imported directly into the `neurotic` namespace. This means that a class like `neurotic.datasets.metadata.MetadataSelector` can be accessed more compactly as `neurotic.MetadataSelector`.

6.1 neurotic.datasets.data

The *neurotic.datasets.data* module implements a function for loading a dataset from selected metadata.

`neurotic.datasets.data.`**`load_dataset`**`(`*metadata*, *blk=None*, *lazy=False*, *signal_group_mode='split-all'*, *filter_events_from_epochs=False*`)`

Load a dataset.

metadata may be a *MetadataSelector* or a simple dictionary containing the appropriate data.

The data_file in metadata is read into a Neo Block using an automatically detected neo.io class if lazy=False or a neo.rawio class if lazy=True. If data_file is unspecified, an empty Neo Block is created instead. If a Neo Block is passed as blk, data_file is ignored.

Epochs and events loaded from annotations_file and epoch_encoder_file and spike trains loaded from tridesclous_file are added to the Neo Block.

If lazy=False, parameters given in metadata are used to apply filters to the signals, to detect spikes using amplitude discriminators, to calculate smoothed firing rates from spike trains, to detect bursts of spikes, and to calculate the rectified area under the curve (RAUC) for each signal.

6.2 neurotic.datasets.download

The *neurotic.datasets.download* module implements a general purpose download function that handles connecting to remote servers, performing authentication, and downloading files with progress reporting. The function handles various errors and will automatically reprompt the user for login credentials if a bad user name or password is given.

The module installs an urllib.request.HTTPBasicAuthHandler and a *neurotic.datasets.ftpauth.FTPBasicAuthHandler* at import time.

neurotic.datasets.download.**download**(*url*, *local_file*, *overwrite_existing=False*, *show_progress=True*, *bytes_per_chunk=8192*)

Download a file.

6.3 neurotic.datasets.ftpauth

The *neurotic.datasets.ftpauth* module implements a urllib.request-compatible FTP handler that prompts for and remembers passwords.

class neurotic.datasets.ftpauth.**FTPBasicAuthHandler**(*password_mgr=None*)

This subclass of urllib.request.FTPHandler implements basic authentication management for FTP connections. Like urllib.request.HTTPBasicAuthHandler, this handler for FTP connections has a password manager that it checks for login credentials before connecting to a server.

This subclass also ensures that file size is included in the response header, which can fail for some FTP servers if the original FTPHandler is used.

This handler can be installed globally in a Python session so that calls to urllib.request.urlopen('ftp://...') will use it automatically:

```
>>> handler = FTPBasicAuthHandler()
>>> handler.add_password(None, uri, user, passwd)  # realm must be None
>>> opener = urllib.request.build_opener(handler)
>>> urllib.request.install_opener(opener)
```

neurotic.datasets.ftpauth.**setup_ftpauth**()

Install *neurotic.datasets.ftpauth.FTPBasicAuthHandler* as the global default FTP handler.

Note that urllib.request.install_opener() used here will remove all other non-default handlers installed in a different opener, such as an urllib.request.HTTPBasicAuthHandler.

6.4 `neurotic.datasets.metadata`

The *`neurotic.datasets.metadata`* module implements a class for reading metadata files.

class `neurotic.datasets.metadata.`**`MetadataSelector`**(*file=None*, *local_data_root=None*, *remote_data_root=None*, *initial_selection=None*)

A class for managing metadata.

A metadata file can be specified at initialization, in which case it is read immediately. The file contents are stored as a dictionary in *`all_metadata`*.

```
>>> metadata = MetadataSelector(file='metadata.yml')
>>> print(metadata.all_metadata)
```

File contents can be reloaded after they have been changed, or after changing `file`, using the *`load()`* method.

```
>>> metadata = MetadataSelector()
>>> metadata.file = 'metadata.yml'
>>> metadata.load()
```

A particular metadata set contained within the file can be selected at initialization with `initial_selection` or later using the *`select()`* method. After making a selection, the selected metadata set is accessible at *`metadata.selected_metadata`*, e.g.

```
>>> metadata = MetadataSelector(file='metadata.yml')
>>> metadata.select('Data Set 5')
>>> print(metadata.selected_metadata['data_file'])
```

A compact indexing method is implemented that allows the selected metadata set to be accessed directly, e.g.

```
>>> print(metadata['data_file'])
```

This allows the MetadataSelector to be passed to functions expecting a simple dictionary corresponding to a single metadata set, and the selected metadata set will be used automatically.

Files associated with the selected metadata set can be downloaded individually or all together, e.g.

```
>>> metadata.download('video_file')
```

or

```
>>> metadata.download_all_data_files()
```

The absolute path to a local file or the full URL to a remote file associated with the selected metadata set can be resolved with the *`abs_path()`* and *`abs_url()`* methods, e.g.

```
>>> print(metadata.abs_path('data_file'))
>>> print(metadata.abs_url('data_file'))
```

`abs_path`(*file*)
Convert the relative path of `file` to an absolute path using `data_dir`.

`abs_url`(*file*)
Convert the relative path of `file` to a full URL using `remote_data_dir`.

`all_metadata = None`
A dictionary containing the entire file contents, set by *`load()`*.

download(*file*, ***kwargs*)
Download a file associated with the selected metadata set.

See `neurotic.datasets.download.download()` for possible keyword arguments.

download_all_data_files(***kwargs*)
Download all files associated with the selected metadata set.

See `neurotic.datasets.download.download()` for possible keyword arguments.

keys
The available metadata keys.

load()
Read the metadata file.

select(*selection*)
Select a metadata set.

selected_metadata
The access point for the selected metadata set.

6.5 neurotic.gui.config

The `neurotic.gui.config` module implements a class for configuring and launching ephyviewer for a loaded dataset.

class `neurotic.gui.config.`**EphyviewerConfigurator**(*metadata*, *blk*, *lazy=False*)
A class for launching ephyviewer for a dataset with configurable viewers.

At initialization, invalid viewers are automatically disabled (e.g., the video viewer is disabled if `video_file` is not given in `metadata`). Viewers can be hidden or shown before launch using the built-in methods. Valid viewer names are:

- `traces`
- `traces_rauc`
- `freqs`
- `spike_trains`
- `traces_rates`
- `epochs`
- `epoch_encoder`
- `video`
- `event_list`
- `data_frame`

`launch_ephyviewer()` is provided for starting a new Qt app and launching the ephyviewer main window all at once. `create_ephyviewer_window()` generates just the ephyviewer window and should be used if there is already a Qt app running.

create_ephyviewer_window(*theme='light'*, *ui_scale='small'*, *support_increased_line_width=False*, *show_datetime=False*, *datetime_format='%Y-%m-%d %H:%M:%S'*)
Load data into each ephyviewer viewer and return the main window.

disable(*name*)
Disable the viewer `name`.

enable(*name*)
Enable the viewer `name`.

hide(*name*)
Hide the viewer `name`.

hide_all()
Hide all viewers.

is_enabled(*name*)
Return whether the viewer `name` is enabled.

is_shown(*name*)
Return whether the viewer `name` is shown.

launch_ephyviewer(*theme='light', ui_scale='small', support_increased_line_width=False, show_datetime=False, datetime_format='%Y-%m-%d %H:%M:%S'*)
Start a Qt app and create an ephyviewer window.

show(*name*)
Show the viewer `name`.

show_all()
Show all viewers.

6.6 neurotic.gui.epochencoder

The *neurotic.gui.epochencoder* module implements a subclass of `ephyviewer.datasource.epochs.WritableEpochSource`.

class neurotic.gui.epochencoder.**NeuroticWritableEpochSource**(*filename, possible_labels, color_labels=None, channel_name='', backup=True*)
A subclass of `ephyviewer.datasource.epochs.WritableEpochSource` for custom CSV column formatting and automatic file backup.

6.7 neurotic.gui.notebook

The *neurotic.gui.notebook* module implements Jupyter notebook widget counterparts for the *MetadataSelector* and the *EphyviewerConfigurator*.

class neurotic.gui.notebook.**MetadataSelectorWidget**(*file=None, local_data_root=None, remote_data_root=None, initial_selection=None*)
Interactive list box for Jupyter notebooks that allows the user to select which metadata set they would like to work with.

```
>>> metadata = MetadataSelectorWidget(file='metadata.yml')
>>> display(metadata)
```

After clicking on an item in the list, the selected metadata set is accessible at `metadata.selected_metadata`, e.g.

```
>>> metadata.selected_metadata['data_file']
```

A compact indexing method is implemented that allows the selected metadata set to be accessed directly, e.g.

```
>>> metadata['data_file']
```

This allows the MetadataSelectorWidget to be passed to functions expecting a simple dictionary corresponding to a single metadata set, and the selected metadata set will be used automatically.

class `neurotic.gui.notebook.`**`EphyviewerConfiguratorWidget`**(*metadata*, *blk*, *lazy=False*)

Interactive button grid for Jupyter notebooks that allows the user to select which ephyviewer viewers they would like to display and then launch ephyviewer.

`disable`(*name*)
Disable the viewer `name`.

`enable`(*name*)
Enable the viewer `name`.

`hide`(*name*)
Hide the viewer `name`.

`show`(*name*)
Show the viewer `name`.

6.8 neurotic.gui.standalone

The *`neurotic.gui.standalone`* module implements the main window of the app.

class `neurotic.gui.standalone.`**`MainWindow`**(*file=None*, *initial_selection=None*, *lazy=True*, *theme='light'*, *ui_scale='small'*, *support_increased_line_width=False*, *show_datetime=False*)

The main window of the app.

6.9 neurotic.scripts

The *`neurotic.scripts`* module handles starting the app from the command line. It also provides a convenience function for quickly launching the app using minimal metadata or an existing Neo `Block`, and another for starting a Jupyter server with an example notebook.

`neurotic.scripts.`**`quick_launch`**(*metadata={}*, *blk=None*, *lazy=True*)

Load data, configure the GUI, and launch the app with one convenient function.

This function allows **neurotic** to be used easily in interactive sessions and scripts. For example, dictionaries can be passed as metadata:

```
>>> metadata = {'data_file': 'data.axgx'}
>>> neurotic.quick_launch(metadata=metadata)
```

An existing Neo `Block` can be passed directly:

```
>>> neurotic.quick_launch(blk=my_neo_block)
```

This function is equivalent to the following:

```
>>> blk = load_dataset(metadata, blk, lazy=lazy)
>>> ephyviewer_config = EphyviewerConfigurator(metadata, blk, lazy=lazy)
>>> ephyviewer_config.show_all()
>>> ephyviewer_config.launch_ephyviewer()
```

`neurotic.scripts.`**`launch_example_notebook`**`()`

Start a Jupyter server and open the example notebook.

6.10 `neurotic._elephant_tools`

The *`neurotic._elephant_tools`* module contains functions and classes copied from the elephant package, which are included for convenience and to eliminate dependency on that package. Original code that builds on this work (e.g., a new kernel) is also contained in this module.

Warning: This module and the functions and classes it contains are not intended to be part of **neurotic**'s public API, so the module name begins with an underscore. This module may be moved, renamed, or removed at a future date and replaced with explicit dependence on the elephant package.

`class` `neurotic._elephant_tools.`**`CausalAlphaKernel`**`(`*sigma*, *invert=False*`)`

This modified version of `elephant.kernels.AlphaKernel` shifts time such that convolution of the kernel with spike trains (as in `elephant.statistics.instantaneous_rate()`) results in alpha functions that begin rising at the spike time, not before. The entire area of the kernel comes after the spike, rather than half before and half after, as with `AlphaKernel`. Consequently, CausalAlphaKernel can be used in causal filters.

The equation used for CausalAlphaKernel is

$$K(t) = \begin{cases} (1/\tau^2)\ t\ \exp{(-t/\tau)}, & t > 0 \\ 0, & t \leq 0 \end{cases}$$

with $\tau = \sigma/\sqrt{2}$, where σ is the parameter passed to the class initializer.

In neuroscience a popular application of kernels is in performing smoothing operations via convolution. In this case, the kernel has the properties of a probability density, i.e., it is positive and normalized to one. Popular choices are the rectangular or Gaussian kernels.

Exponential and alpha kernels may also be used to represent the postynaptic current / potentials in a linear (current-based) model.

sigma [Quantity scalar] Standard deviation of the kernel.

invert: bool, optional If true, asymmetric kernels (e.g., exponential or alpha kernels) are inverted along the time axis. Default: False

CHAPTER 7

Release Notes

7.1 neurotic 1.4.0

2020-03-04

7.1.1 New features

- Add computation of smoothed firing rates for spike trains (#189)
- Add `quick_launch` function for easier scripting and interactive sessions (#191, #196)
- Add `blk` param to `load_dataset` for using existing Neo Block (#196)
- Add `rec_datetime` metadata parameter for setting real-world start time (#190)
- Add `past_fraction` metadata parameter for controlling the placement of the vertical line marking the current time (#220)
- Add color options for signal plots to metadata (#177)
- Add options for scaling the user interface (#222, #241, #242)
- Add launch via double click in standalone app (#218)

7.1.2 Help & feedback

- Display loading indicator when launching in the standalone app (#223)
- Prompt the user to reload metadata after using the menu to edit metadata (#226)
- Show "Downloads complete" in app status bar and in log (#219)
- Add menu action status tips to standalone app (#221, #240)
- Add "Check for updates" to help menu (#201)
- Add Help menu actions for opening common URLs (#198, #230)

- Add Python version and **neurotic** install path to About window (#181)
- Display common error messages to status bar in standalone app (#225)
- Display a splash screen when first starting the standalone app (#224)

7.1.3 Jupyter tutorial

Start the tutorial using `neurotic --launch-example-notebook` or view it here.

- Add API tutorial to example Jupyter notebook (#234, #236)
- Add shell commands to example Jupyter notebook for installing neurotic (#185)

7.1.4 Other changes

- Move RAUC calculations to `load_dataset` and store them as annotations (#188)
- Disable RAUC calculations by default (#193)
- Allow `data_file` to be unspecified in metadata and signals to be optional (#195)
- Ignore `video_file` if PyAV is not installed (#231)
- Rename elephant functions (#183)

7.1.5 Bug fixes

- Fix crash when `plots` list is empty (#217)
- Fix plotted signal sampling rate and start time if units are not Hz and sec, respectively (#194)
- Fix incompatibility with old versions of tqdm and unknown download size (#184)

7.1.6 Documentation

- Improve installation and updating documentation (#180, #229)

7.1.7 Logging

- Add package logger and permanent log file (#174)
- Log fatal errors in standalone app instead of crashing (#176)
- Add CLI and GUI options for enabling debug log messages (#178)
- Raise the default threshold for PyAV messages from warning to critical (#179)

7.2 neurotic 1.3.0

2020-01-07

7.2.1 Improvements

- Add burst detection via firing rate thresholding (#156)
- Add button for auto-scaling signals to main window (#150)
- Add metadata color parameters for `amplitude_discriminators` (#166)
- Add metadata parameters `rauc_baseline` and `rauc_bin_duration` (#151)
- Make `data_dir` default to metadata file directory (#163)

7.2.2 Bug fixes

- Unmask FileNotFoundError when local data file is missing (#154)

7.2.3 Documentation

- Add Zenodo archive badge (#162)

7.3 neurotic 1.2.1

2019-12-09

7.3.1 Bug fixes

- Fix loading using Neo IOs lacking `signals_group_mode` (`TypeError: read_block() got an unexpected keyword argument 'signal_group_mode'`) (#143)

7.4 neurotic 1.2.0

2019-12-06

neurotic should now have broader compatibility with file types supported by Neo's `neo.io` classes thanks to two new metadata parameters: `io_class` and `io_args`. See *Data Reader (Neo) Settings* for details.

neurotic is now available on conda-forge! See *Alternate Method: conda (recommended for Pythonistas)* for details on how to install.

7.4.1 Improvements

- Add metadata parameters `io_class` and `io_args` (#137)

7.4.2 Documentation

- Add conda-forge installation instructions (#128)

7.5 neurotic 1.1.1

2019-10-17

7.5.1 Changes

- Remove elephant as a dependency (#120)

7.6 neurotic 1.1.0

2019-10-09

7.6.1 Improvements

- Add API parameters and CLI argument for displaying the real-world date and time (potentially inaccurate) (#110, #118)
- Add printer-friendly theme (white background) (#114)

7.6.2 Documentation

- Add documentation of GIN URLs for public and private repos (#113)

7.7 neurotic 1.0.0

2019-07-27

First stable release!

7.7.1 Improvements

- Major API changes (#104, #100, #106)
  - In preparation for this stable release, many formerly public classes and functions were made private. This was done to minimize the number of public classes/functions, which beginning with this release will be treated as stable APIs that are ideally modified only in backwards compatible ways. Users should trust that public classes and functions will not be changed without good reason and a major version bump.
- Many improvements to the documentation, including the addition of an API Reference Guide
- Add example Jupyter notebook and command line argument for launching it (#108)
- Add file overwrite option to download functions (#106)
- Reserve the metadata keyword `neurotic_config` for global parameters (#93)
  - The `remote_data_root` key must now be nested under `neurotic_config`.

7.7.2 Bug fixes

- Fix crash when epoch encoder file contains labels not listed in metadata (#97)
- Allow amplitude discriminators to be specified with arbitrary units (#99)

7.8 neurotic 0.7.0

2019-07-21

7.8.1 Improvements

- New documentation hosted at Read the Docs: https://neurotic.readthedocs.io
- Add menu action for opening metadata in editor (#83)
- Add menu action for opening the selected data directory (#84)
- Add list of installed versions of dependencies and doc links to About window (#44, #65)

7.8.2 Bug fixes

- Fix files remaining locked after closing a fast-loaded window (#69)
- Fix launching from command line with bad metadata argument (#82)

7.9 neurotic 0.6.0

2019-07-10

7.9.1 Improvements

- Add a basic "About neurotic" window with version and website information (#38)
- Update logo (#39)
- Add keywords and project URLs to package metadata (#40)

7.10 neurotic 0.5.1

2019-07-09

7.10.1 Compatibility updates

- Compatibility update for RawIOs with non-zero offset (#37)

7.11 neurotic 0.5.0

2019-07-06

7.11.1 Improvements

- Support fast (lazy) loading in Neo < 0.8.0 (#35)
- Add "git." and conditionally ".dirty" to dev local version identifier (#34)

7.12 neurotic 0.4.2

2019-07-06

7.12.1 Bug fixes

- Fix for EstimateVideoJumpTimes regression introduced in 0.4.0 (#33)

7.13 neurotic 0.4.1

2019-07-02

7.13.1 Compatibility updates

- Change sources of development versions of dependencies (#32)
- Compatibility update for scaling of raw signals (#31)

7.14 neurotic 0.4.0

2019-07-01

7.14.1 Improvements

- Show epochs imported from CSV files with zero duration in epoch viewer (#27)
- Show epochs/events imported from data file in events list/epoch viewer (#28)
- Alphabetize epoch and event channels by name (#29)

7.15 neurotic 0.3.0

2019-06-29

7.15.1 Improvements

- Remove dependency on ipywidgets by making notebook widgets optional (#25)
  - Notebook widget classes renamed: `MetadataSelector` → `MetadataSelectorWidget`, `EphyviewerConfigurator` → `EphyviewerConfiguratorWidget`
- Add app description and screenshot to README (#22)
- Promote to beta status (#23)

7.16 neurotic 0.2.0

2019-06-28

7.16.1 Improvements

- Add basic command line arguments (#14)
- Add continuous integration with Travis CI for automated testing (#13)
- Add some tests (#15, #16)
- Migrate example data to GIN (#18)

7.16.2 Bug fixes

- Fix crash when downloading from a server that does not report file size (#17)
- Raise an exception if a Neo RawIO cannot be found for the data file (#12)

7.17 neurotic 0.1.1

2019-06-22

7.17.1 Bug fixes

- Fix various downloader errors (#7)

7.18 neurotic 0.1.0

2019-06-22

- First release

Python Module Index

Index