Psychopy Tools Documentation

Release 0.1.0

Eshin Jolly

Feb 20, 2018

Contents

This package contains some tools to make working with Psychopy easier.

CHAPTER 1

Example functionality

- Graceful clean-up function to handle quitting an experiment safely by closing devices (e.g. serial port) and data files
- Extra convenience methods for `core.Clock()` that automatically do non-slip timing (e.g. for fMRI studies)
- Extra `.draw_feature()` methods for visual elements that alter functionality (e.g. drawing/moving a `visual.RatingScale` without collecting a rating)

CHAPTER 2

Installation

```
pip install git+https://github.com/ejolly/psychopy_tools
```

(*coming to pypi soon!*)

2.1 Usage

2.1.1 1. Add to Psychopy's path

If using the standalone Psychopy application you'll need to add this to Psychopy's path:

- Find out where `pip` installed the package (search for your `site-packages`) directory, it'll be inside
- If you're using `conda` this should be somewhere like: `/Users/You/anaconda/lib/python2.7/site-packages/psychopy_tools-0.1.0-py2.7.egg` (*note: the version number maybe different!*)
- Copy this path
- In Psychopy goto: File > Preferences > General
- Paste the path into the "paths" field, e.g `['/Users/You/anaconda/lib/python2.7/site-packages/psychopy_tools-0.1.0-py2.7.egg']`
- Click OK to apply

2.1.2 2. Import within your experiment script

You can just import this as if you would any other Psychopy functionality:

```
from psychopy.presentation import clean_up
```

2.1.3 3. Typical use cases

Generally Psychopy Tools contains modules that either:

1. work like normal Python functions
2. are designed to augment existing Psychopy classes with new abilities.
3. are subclasses of Psychopy classes with added attributes or functionality

An example of 1. is the `random_jitter` function which can be used like any Python function. For example creating a random sequence of 100 ITIs (jitters) with a mean of 5s, a min of 4s and a max of 20s:

```
from psychopy_tools.stim_gen import random_jitter

ITIs = random_jitter(desired_mean=5,
                     num_trials=200,
                     min_iti=4,
                     max_iti=20)
```

An example of 2. is the `wait_time` function which is a convenience method designed to give `clock.Clock()` the ability to wait for a specific amount of time or run a function for a specific amount of time, using non-slip timing. To use a function like this you must first create the object you want (in this case a clock) and then assign this function to it as a new method. For example:

```
from psychopy_tools.presentation import wait_time

# We need a Python function that will convert wait_time to a method
from types import MethodType

# Then create your clock as your normally would
timer = core.Clock()

# Then add wait_time as a new method to the instance by wrapping it
timer.wait_time = MethodType(wait_time,timer)

# Use just like you would use any other timer method
timer.wait_time(5) # just wait 5 seconds
```

An example of 3. is the `rating.RatingScale` class which sub-classes Psychopy's *visual.RatingScale*. This can be imported and used anywhere in a script one would normally use *visual.RatingScale*. It behaves identically except for a few differences: a) the ability to take an optional *bounds* parameter which prevents a user from making responses above or below certain values. This should be provided as a list `bounds = [lower, upper]` in units of the scale. It's left to the user to visualize these bounds in the window (b) a `.draw_only()` method which only displays the scale without listening for responses. This can be useful for using a `RatingScale` as a stimulus as oppose to an input controller, for example when visualizing values on a `RatingScale` or controlling the display of one `RatingScale` from another that actually takes inputs.

See the *Function Help* for complete info on how to use specific functions.

2.2 Function Help

This reference provides detailed documentation for all the functions in Psychopy Tools.

2.2.1 psychopy_tools.presentation: Presentation Functions

Functions typically used when running the presentation script.

psychopy_tools.presentation.**clean_up**(*window*, *serial=None*, *labjack=None*, *data_files=None*)

Safer close function on premature experiment quit designed to gracefully close datafiles and devices. Minimum requires a window to close, but can also close: a serial object (typically one created to collect fMRI triggers), a labjack object (typically one created to trigger biopac/psychophys trigger), and a list of data_files (typically opened with Python's 'open').

Note: Sometimes psychopy will issue a crash message when prematurely quitting. This is nothing to worry about.

Parameters

- **window** – window handle from psychopy script
- **serial** – scanner trigger serial object instance
- **labjack** – labjack (psychophys) object instance
- **data_files** – list of data files

Examples

This function is designed to be used as Psychopy global key function for example:

```
>>> from psychopy_tools.presentation import clean_up
```

Then later in your script after window creation:

```
>>> # Close window only
>>> event.globalKeys.add(key='q',func=clean_up, name='shutdown',func_
↪args=(window))
```

Or embedded within a while loop:

```
>>> while True:
>>>     # do something
>>>
>>>     # But be able to quit while in the loop
>>>     if len(event.getKeys(['q'])):
>>>         # Close window, devices and data files
>>>         clean_up(window,scanner,biopac,data_files)
```

psychopy_tools.presentation.**draw_scale_only**(*self*)

DEPRECATED Use *psychopy_tools.rating.RatingScale* which has this method built-in.

Convenience method to only draw a visual.RatingScale scale, but not collect a rating.

Examples

Use this by augmenting an already existing Psychopy visual.RatingScale instance:

```
>>> # First we need a Python function to make this "method-able"
>>> from types import MethodType
>>>
```

```
>>> # Then create your scale as you normally would
>>> myScale = visual.RatingScale(...)
>>>
>>> # Then add this as a new method to the instance
>>> myScale.draw_only = MethodType(draw_scale_only,myScale)
>>>
>>> # Use it just like you would have used .draw()
>>> myScale.draw_only()
```

psychopy_tools.presentation.**wait_for_click**(*self*)

Similar to event.waitKeys(), but waits for a mouse click. Specifically waits for the press **and** release of a mouse click. Agnostic to which mouse button was clicked.

Examples

Use this by augmenting an already existing Psychopy event.Mouse instance:

```
>>> # First we need a Python function to make this "method-able"
>>> from types import MethodType
>>>
>>> # Then create your scale as you normally would
>>> myMouse = event.Mouse(...)
>>>
>>> # Then add this as a new method to the instance
>>> myMouse.wait_for_click = MethodType(wait_for_click,myMouse)
>>>
>>> # Use it just like you would have used any other method
>>> myMouse.wait_for_click()
```

psychopy_tools.presentation.**wait_time**(*self*, *duration*, *func=None*, *func_args=None*)

Convenience method to augment core.Clock with non-slip timing. Can just wait a specific duration and do nothing, or run some function (e.g. get a rating)

Parameters

- **duration** (*int/float*) – time to wait in seconds
- **func** (*function handle*) – function to execute for the duration

Examples

Use this by augmenting an already existing Psychopy core.Clock instance:

```
>>> # First we need a Python function to make this "method-able"
>>> from types import MethodType
>>>
>>> # Then create your clock as your normally would
>>> timer = core.Clock()
>>>
>>> # Then add this as a new method to the instance
>>> timer.wait_time = MethodType(wait_time,timer)
>>>
>>> # Use just like you would use any other timer method
>>> timer.wait_time(5) # just wait 5 seconds
>>>
>>> # You can also present something for a specific amount of time or run any␣
↪arbitrary function
```

```
>>>
>>> time.wait_time(5,my_func,func_args) #run my_func(func_args) in a while loop␣
↪for 5 seconds
```

2.2.2 psychopy_tools.stim_gen: Stimulus Generation Functions

Functions associated with stimulus generation and randomization.

psychopy_tools.stim_gen.**random_jitter**(*num_trials*, *desired_mean=6*, *iti_min=1*, *iti_max=None*, *discrete=True*, *tolerance=0.05*, *nsim=20000*, *plot=True*)

Create a series of ITIs (in seconds) with a given mean, number of trials and optionally minimum and maximum. ITIs follow either a discrete geometric distribution or a continuous exponential based on user settings. Either way produces a sequence with more shorter ITIs and fewer longer ITIs.

Perhaps better for fast event-related designs where a mean ITI is desired that comes from a skewed distribution with many shorter ITIs and a few long ITIs.

NOTE: The min ITI is guaranteed, but the max ITI is an upper bound. This means the generated maximum may actually be lower than the desired max. Setting the maximum too low will result in harder/no solutions and will cause the distribution to be less well behaved.

Parameters

- **num_trials** (*int*) – number of trials (number of ITIs to create)
- **desired_mean** (*float*) – desired mean ITI; default 6
- **iti_min** (*int/float*) – minimum ITI length; guaranteed; default 1
- **iti_max** (*int/float*) – maximum ITI length; only guaranteed that ITIs will not be longer than this; default None
- **discrete** (*bool*) – should ITIs be integers only (discrete geometric) or floats (continuous exponential); default discrete
- **tolerance** (*float*) – acceptable difference from desired mean; default 0.05
- **nsim** (*int*) – number of search iterations; default 10,000
- **plot** (*bool*) – plot the distribution for visual inspection; default True

Returns sequence

Return type seq (np.ndarray)

psychopy_tools.stim_gen.**random_uniform_jitter**(*num_trials*, *desired_mean=6*, *iti_min=2*, *iti_max=None*, *discrete=True*, *tolerance=0.05*, *nsim=20000*, *plot=True*)

Create a series of ITIs (in seconds) from a uniform distribution. You must provid any **two** of the following three inputs: desired_mean, iti_min, iti_max. This is because for uniform sampling, the third parameter is necessarily constrained by the first two.

Perhaps useful for slow event related designs, where a longish mean ITI is desired with some variability around that mean.

ITIs follow either a discrete or continuous uniform distribution. If a small amount of variability is desired around a particular mean, simply provide a desired_mean, and a iti_min (or iti_max) that is very close to desired_mean.

Parameters

- **num_trials** (*int*) – number of trials (number of ITIs to create)

- **desired_mean** (*float*) – desired mean ITI; default 6
- **iti_min** (*int/float*) – minimum ITI length; guaranteed; default 1
- **iti_max** (*int/float*) – maximum ITI length; guaranteed; default 10 (computed)
- **discrete** (*bool*) – should ITIs be integers only (discrete) or floats (continuous); default discrete
- **tolerance** (*float*) – acceptable difference from desired mean; default 0.05
- **nsim** (*int*) – number of search iterations; default 20,000
- **plot** (*bool*) – plot the distribution for visual inspection; default True

Returns sequence

Return type seq (np.ndarray)

2.2.3 psychopy_tools.rating: Visual Rating Scale

psychopy_tools.rating.**RatingScale**

CHAPTER 3

Usage

CHAPTER 4

Function Help

4.1 Credits

This package was created with Cookiecutter and the ejolly/cookiecutter-pypackage template.

Python Module Index

p

Index