pphelper Documentation

Release 0.6

Richard Höchenberger

September 01, 2015

Contents

1 About pphelper **1**

2 The `racemodel` Module **3**
2.1 Provides 3

3 The `utils` Module **11**
3.1 Provides 11

4 The `hardware` Module **13**
4.1 Olfactometer 13
4.2 AnalogInput 14
4.3 Gustometer 15

5 The `image` Module **17**
5.1 pphelper.image 17

6 The `utils` Module **19**
6.1 Provides 19

Python Module Index **21**

CHAPTER 1

About pphelper

pphelper is a collection of tools that can be used for the analysis of psychophysical data.

CHAPTER 2

The `racemodel` Module

Race model inequality analysis implementation, based on Ulrich, Miller, and Schröter (2007): 'Testing the race model inequality: An algorithm and computer programs', published in Behavior Research Methods 39 (2), pp. 291-302.

2.1 Provides

- `gen_cdf` : Estimate the cumulative distribution function from response time data.
- `gen_cdfs_from_list` : Convenience function: Applys `gen_cdf` to a list of data sets.
- `gen_percentiles` : Calculate equally spaced percentiles values.
- `get_percentiles_from_cdf` : Get the values (response times) of a cumulative distribution function at the specified percentiles.
- `gen_step_fun` : Generate a step function from a set of observed response times.

`pphelper.racemodel.`**`gen_cdf`**(*rts*, *t_max=None*)

Estimate the cumulative frequency polygon from response time data.

Parameters

- **rts** (*array_like*) – The raw response time data. Data does not need to be ordered and may contain duplicate values.
- **t_max** (*int, optional*) – Up to which time point (in milliseconds) the model should be calculated. If not specified, the maximum value of the supplied input data will be used.

Returns A Series containing the estimated cumulative frequency polygon, indexed by the time points in ms.

Return type DataFrame or Series

See also:

`gen_cdfs_from_dataframe()`, `gen_cdfs_from_list()`, `get_percentiles_from_cdf()`, `gen_step_fun()`

Notes

Response times will be rounded to 1 millisecond. The algorithm is heavily adapted from the one described by Ulrich, Miller, and Schröter (2007): 'Testing the race model inequality: An algorithm and computer programs', published in Behavior Research Methods 39 (2), pp. 291-302.

Examples

```
>>> from pphelper.racemodel import gen_cdf
>>> import numpy as np
>>> RTs = np.array([234, 238, 240, 240, 243, 243, 245, 251, 254, 256, 259, 270, 280])
>>> gen_cdf(RTs, t_max=RTs.max())
t
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    0
12    0
13    0
14    0
...
266    0.856643
267    0.863636
268    0.870629
269    0.877622
270    0.884615
271    0.892308
272    0.900000
273    0.907692
274    0.915385
275    0.923077
276    0.930769
277    0.938462
278    0.946154
279    0.953846
280    1.000000
Length: 281, dtype: float64
```

`pphelper.racemodel.`**`gen_cdfs_from_dataframe`**(*data*, *rt_column=u'RT'*, *modality_column=u'Modality'*, *names=None*)

Create cumulative distribution functions (CDFs) for response time data.

Parameters

- **data** (*DataFrame*) – A DataFrame with containing at least two columns: one with response times, and another one specifying the corresponding modalities.
- **rt_column** (*string, optional*) – The name of the column containing the response times. Defaults to `RT`.
- **modality_column** (*string, optional*) – The name of the column containing the modalities corresponding to the response times. Defaults to `Modality`.
- **names** (*list, optional*) – A list of length 4, supplying the names of the modalities. The first three elements specify the modalities in the input data to consider. These three and the fourth argument are also used to label the columns in the returned DataFrame. If this argument is not supplied, a default list `['A', 'B', 'AB']` will be used.

Returns **results** – A DataFrame containing the empirical cumulative distribution functions generated from the input, one CDF per column. The number of columns depends on the number of unique values in the *modality_column* or on the *names* argument,

Return type DataFrame

See also:

`gen_cdf()`, `gen_cdfs_from_list()`, `gen_step_fun()`, `get_percentiles_from_cdf()`

Notes

This function internally calls `gen_cdf`. Please see this function to find out about additional optional keyword arguments.

Examples

```
>>> from pphelper.racemodel import gen_cdfs_from_dataframe
>>> import pandas as pd
>>> import numpy as np
>>> data = pd.DataFrame({'RT': np.array([244, 249, 257, 260, 264, 268, 271, 274, 277, 291,
... 245, 246, 248, 250, 251, 252, 253, 254, 255, 259, 263, 265, 279, 282, 284, 319,
... 234, 238, 240, 240, 243, 243, 245, 251, 254, 256, 259, 270, 280]),
... 'Modality': ['x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x',
... 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y',
... 'z', 'z', 'z', 'z', 'z', 'z', 'z', 'z', 'z', 'z', 'z', 'z', 'z', ]})
>>> gen_cdfs_from_dataframe(data)
            x         y  z
t
0    0.000000  0.000000  0
1    0.000000  0.000000  0
2    0.000000  0.000000  0
3    0.000000  0.000000  0
4    0.000000  0.000000  0
5    0.000000  0.000000  0
6    0.000000  0.000000  0
7    0.000000  0.000000  0
8    0.000000  0.000000  0
9    0.000000  0.000000  0
10   0.000000  0.000000  0
11   0.000000  0.000000  0
12   0.000000  0.000000  0
13   0.000000  0.000000  0
14   0.000000  0.000000  0
15   0.000000  0.000000  0
16   0.000000  0.000000  0
17   0.000000  0.000000  0
18   0.000000  0.000000  0
19   0.000000  0.000000  0
20   0.000000  0.000000  0
21   0.000000  0.000000  0
22   0.000000  0.000000  0
23   0.000000  0.000000  0
24   0.000000  0.000000  0
25   0.000000  0.000000  0
26   0.000000  0.000000  0
27   0.000000  0.000000  0
```

```
28   0.000000  0.000000  0
29   0.000000  0.000000  0
..        ...       ... ..
290  0.942857  0.916964  1
291  1.000000  0.918750  1
292  1.000000  0.920536  1
293  1.000000  0.922321  1
294  1.000000  0.924107  1
295  1.000000  0.925893  1
296  1.000000  0.927679  1
297  1.000000  0.929464  1
298  1.000000  0.931250  1
299  1.000000  0.933036  1
300  1.000000  0.934821  1
301  1.000000  0.936607  1
302  1.000000  0.938393  1
303  1.000000  0.940179  1
304  1.000000  0.941964  1
305  1.000000  0.943750  1
306  1.000000  0.945536  1
307  1.000000  0.947321  1
308  1.000000  0.949107  1
309  1.000000  0.950893  1
310  1.000000  0.952679  1
311  1.000000  0.954464  1
312  1.000000  0.956250  1
313  1.000000  0.958036  1
314  1.000000  0.959821  1
315  1.000000  0.961607  1
316  1.000000  0.963393  1
317  1.000000  0.965179  1
318  1.000000  0.966964  1
319  1.000000  1.000000  1
```

[320 rows x 3 columns]

`pphelper.racemodel.`**`gen_cdfs_from_list`**(*data*, *t_max=None*, *names=None*, *return_type=u'dataframe'*)

Estimate the empirical CDFs for a list of arrays.

The is a convenience function that wraps `gen_cdf`.

Parameters

- **data** (*list of array_like objects*) – A list of raw response time arrays. The RTs do not have to be ordered and may contain duplicate values.
- **t_max** (*int, optional*) – Up to which time point (in milliseconds) the model should be calculated. If not specified, the maximum value of the supplied input data will be used.
- **return_type** (*{'dataframe', 'list'}*) – The format of the returned object. *dataframe* returns a DataFrame, *list* returns a list of *Series*.

Returns The estimated empirical CDFs as columns of a DataFrame (default) or as a list of Series (if *return_type='list'*).

Return type DataFrame or list of Series

Raises `ValueError` – If the *name* parameter does not have the same lengths as the data list.

See also:

gen_cdf(), gen_cdfs_from_dataframe(), get_percentiles_from_cdf(), gen_step_fun()

Examples

```
>>> from pphelper.racemodel import gen_cdfs_from_list
>>> import numpy as np
>>> RTs = [np.array([234, 238, 240, 240, 243, 243, 245, 251, 254, 256, 259, 270,
 280]), np.array([244, 249, 257, 260, 264, 268, 271, 274, 277, 291])]
>>> gen_cdfs_from_list(RTs, names=['CondA', 'CondB'])
        CondA     CondB
t
0    0.000000  0.000000
1    0.000000  0.000000
2    0.000000  0.000000
3    0.000000  0.000000
4    0.000000  0.000000
5    0.000000  0.000000
6    0.000000  0.000000
7    0.000000  0.000000
8    0.000000  0.000000
9    0.000000  0.000000
10   0.000000  0.000000
11   0.000000  0.000000
12   0.000000  0.000000
13   0.000000  0.000000
14   0.000000  0.000000
15   0.000000  0.000000
16   0.000000  0.000000
17   0.000000  0.000000
18   0.000000  0.000000
19   0.000000  0.000000
20   0.000000  0.000000
21   0.000000  0.000000
22   0.000000  0.000000
23   0.000000  0.000000
24   0.000000  0.000000
25   0.000000  0.000000
26   0.000000  0.000000
27   0.000000  0.000000
28   0.000000  0.000000
29   0.000000  0.000000
..        ...       ...
262  0.828671  0.400000
263  0.835664  0.425000
264  0.842657  0.450000
265  0.849650  0.475000
266  0.856643  0.500000
267  0.863636  0.525000
268  0.870629  0.550000
269  0.877622  0.583333
270  0.884615  0.616667
271  0.892308  0.650000
272  0.900000  0.683333
273  0.907692  0.716667
274  0.915385  0.750000
275  0.923077  0.783333
```

```
276  0.930769  0.816667
277  0.938462  0.850000
278  0.946154  0.857143
279  0.953846  0.864286
280  1.000000  0.871429
281  1.000000  0.878571
282  1.000000  0.885714
283  1.000000  0.892857
284  1.000000  0.900000
285  1.000000  0.907143
286  1.000000  0.914286
287  1.000000  0.921429
288  1.000000  0.928571
289  1.000000  0.935714
290  1.000000  0.942857
291  1.000000  1.000000
```

[292 rows x 2 columns]

`pphelper.racemodel.`**`gen_percentiles`**(*n=10*)

Calculate n equally spaced percentiles.

Parameters **n** (*int, optional*) – The number of percentiles to generate. Defaults to 10. Floats will be rounded.

Returns **p** – 1-dimensional array of the calculated percentiles.

Return type ndarray

Raises `TypeError` – If the supplied percentile number could not be converted to a rounded integer.

See also:

`get_percentiles_from_cdf()`

Examples

```
>>> from pphelper.racemodel import gen_percentiles
>>> gen_percentiles()
array([ 0.05,  0.15,  0.25,  0.35,  0.45,  0.55,  0.65,  0.75,  0.85,  0.95])
```

`pphelper.racemodel.`**`gen_step_fun`**(*rts*)

Generate a step function from an observed response time distribution.

Parameters **rts** (*array_like*) – The input data (usually response times) to generate a step function from. Does not have to be ordered and may contain duplicates.

Returns A Series of the ordered response times (smallest to largest), indexed by their respective percentiles.

Return type Series

See also:

`gen_cdf()`, `gen_cdfs_from_dataframe()`, `gen_cdfs_from_list()`

Examples

```
>>> from pphelper.racemodel import gen_step_fun
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> RTs = np.array([234, 238, 240, 240, 243, 243, 245, 251, 254, 256, 259, 270, 280])
>>> sf = gen_step_fun(RTs)
>>> plt.step(sf, sf.index, where='post'); plt.show()
```

pphelper.racemodel.**get_percentiles_from_cdf**(*cdf*, *p=None*, *num_p=10*, *time_index=u't'*)

Interpolate the percentile boundaries.

Parameters

- **cdf** (*Series*) – The cumulative distribution polygon. Usually generated by *gen_cdf()*.
- **p** (*array_like, optional*) – The percentiles for which to get values from the polygon. Usually generated by *gen_percentiles()*. If this is supplied, the *num_p* argument will be ignored.
- **num_p** (*int, optional*) – The number of equally spaced percentiles to generate. Will be ignored if *p* is supplied. Defaults to 10.
- **time_index** (*str, optional*) – The name of the index storing the time (in milliseconds). This will only be used if the supplied CDF is a pandas *Series* with a *MultiIndex*. Defaults to *t*.

Returns Returns a Series of interpolated percentile boundaries (fictive response times).

Return type Series

Raises `TypeError` – If the supplied percentile object could not be cast into an array, or if the CDF object is not a Series.

See also:

`gen_cdf()`, `gen_cdfs_from_list()`, `gen_cdfs_from_dataframe()`, `gen_percentiles()`

Examples

```
>>> from pphelper.racemodel import gen_cdf, gen_percentiles, get_percentiles_from_cdf
>>> import numpy as np
>>> RTs = np.array([234, 238, 240, 240, 243, 243, 245, 251, 254, 256, 259, 270, 280])
>>> cdf = gen_cdf(RTs)
>>> percentiles = gen_percentiles(5)
>>> get_percentiles_from_cdf(cdf, percentiles)
p
0.1    237.20
0.3    241.35
0.5    245.00
0.7    255.20
0.9    272.00
dtype: float64
```

pphelper.racemodel.**sum_cdfs**(*cdfs*)

Calculate the sum of multiple cumulative distribution functions.

Parameters **cdfs** (*list*) – A list of CDFs generated with `gen_cdf`, `gen_cdfs_from_list`, or `gen_cdfs_from_dataframe`.

Returns The sum of the CDFs in the interval [0, 1], indexed by the time in milliseconds.

Return type Series

Raises

- `ValueError` – If the supplied CDFs have unequal lengths.
- `IndexError` – If the indices of the supplied CDF Series objects do not match.

Notes

First calculates the sum of the CDFs, and returns the element-wise minima *min[(sum, 1)*.

See also:

`gen_cdf()`, `gen_cdfs_from_dataframe()`, `gen_cdfs_from_list()`

Examples

```
>>> from pphelper.racemodel import gen_cdfs_from_list, sum_cdfs
>>> import numpy as np
>>> RTs = [np.array([234, 238, 240, 240, 243, 243, 245, 251, 254, 256, 259, 270, 280]), np.array
>>> cdfs = gen_cdfs_from_list(RTs, names=['A', 'B'])
>>> sum_cdfs([cdfs['A'], cdfs['B']])
t
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    0
12    0
13    0
14    0
...
277    1
278    1
279    1
280    1
281    1
282    1
283    1
284    1
285    1
286    1
287    1
288    1
289    1
290    1
291    1
Length: 292, dtype: float64
```

CHAPTER 3

The `utils` Module

Some functions that come in handy when working with psychophysics datasets.

3.1 Provides

- `d_prime` : Calculate the sensitivity index d' ("d-prime").
- `criterion` : Calculate the decision criterion C.

`pphelper.sdt.`**`a_prime`**(*hits*, *false_alarms*, *n*, *nafc=1*)

Calculate the sensitivity index A'.

Parameters

- **hits** (*float*) – The number of hits when detecting a signal.
- **false_alarms** (*float*) – The number of false alarms.
- **n** (*int*) – The number of trials in target and no-target trials.
- **nafc** (*int, optional*) – The number of alternative choices in the task. A value of `1` implies a Yes/No task. Defaults to 1.

Returns **A** – The calculated A'.

Return type float

Example

```
>>> from pphelper import sdt
>>> sdt.A_prime(20, 10, 25)
0.79166666666666674
```

`pphelper.sdt.`**`criterion`**(*hits*, *false_alarms*, *n*, *nafc=1*)

Calculate the decision criterion C.

Parameters

- **hits** (*float*) – The number of hits when detecting a signal.
- **false_alarms** (*float*) – The number of false alarms.
- **n** (*int*) – The number of trials in target and no-target trials.

- **nafc** (*int, optional*) – The number of alternative choices in the task. A value of `1` implies a Yes/No task. Defaults to 1.

Returns **C** – The decision criterion. This will be zero for an unbiased observer, and non-zero otherwise. In a 1-AFC (Yes/No) task, a value smaller than 0 implies a bias to responding "Yes", and a value greater than 0 a bias to responding "No".

Return type float

Example

```
>>> from pphelper import sdt
>>> sdt.criterion(20, 10, 25)
-0.29413706521855731
```

`pphelper.sdt.`**`d_prime`**(*hits*, *false_alarms*, *n*, *nafc=1*)

Calculate the sensitivity index d' ("d-prime").

Parameters

- **hits** (*float*) – The number of hits when detecting a signal.
- **false_alarms** (*float*) – The number of false alarms.
- **n** (*int*) – The number of trials in target and no-target trials.
- **nafc** (*int, optional*) – The number of alternative choices in the task. A value of `1` implies a Yes/No task. Defaults to 1.

Returns **d** – The calculated d' value, z(hit_rate) - z(fa_rate).

Return type float

Example

```
>>> from pphelper import sdt
>>> sdt.d_prime(20, 10, 25)
1.094968336708714
```

CHAPTER 4

The `hardware` Module

4.1 Olfactometer

class `pphelper.hardware.`**`Olfactometer`**(*ni_lines=u'Dev1/port0/line0:7'*, *ni_trigger_line=None*, *ni_task_name=u'Olfactometer'*, *use_threads=True*)

Bases: `pphelper.hardware._StimulationApparatus`

Provides an interface to the ValveLink devices used to control olfactometers.

`add_stimulus`(*name*, *bitmask*, *duration=1*, *bitmask_offset=None*, *trigger_time=None*, *replace=False*, ***kwargs*)

Add a stimulus to the stimulus set of this apparatus.

Parameters

- **name** (*string*) – A unique identifier of the stimulus to add.
- **bitmask** (*array_like*) – The bitmask specifying the valve positions required to present this stimulus.
- **duration** (*float, optional*) – The duration of the stimulation, specified in seconds. Defaults to 1 second.
- **bitmask_offset** (*array_like, optional*) – The bitmask specifying the valve positions after the stimulation has finished. If not specified, all valves will be closed at the end of the stimulation.
- **trigger_time** (*float, optional*) – The time (in terms of the `psychopy.core.getTime` timebase) at which the stimulation should be triggered. If `None`, trigger immediately. Defaults to `None`.
- **replace** (*bool, optional*) – Whether an already existing stimulus of the same name should be replaced or not. Defaults to `False`.

Notes

Any additional keyword arguments will be added as additional stimulus properties.

See also:

`remove_stimulus(), select_stimulus(), stimulate()`

`remove_stimulus`(*name*)

Remove the specified stimulus, identified by its unique name, from the stimulus list.

Parameters **name** (*string*) – The unique name of the stimulus to remove.

select_stimulus(*name*)

Select the specified stimulus for the next stimulation.

Parameters **name** (*string*) – The unique name of the stimulus to select.

stimulate(*blocking_wait=False*)

Start the stimulation with the currently selected stimulus.

The trigger pulse will open the valves. They are then left open for the intended duration of the stimulus. After that, they will be switched to the offset state specified by the stimulus.

Parameters **blocking_wait** (*bool, optional*) – Specifies whether the stimulation thread should be *joined* or not, i.e. whether we should wait for it to finish (blocking other operations), or return immediately. This parameter will be ignored if threads are not used for stimulation. Defaults to *False*, i.e. non-blocking behavior.

See also:

`add_stimulus()`, `select_stimulus()`

Notes

`stimulate` invokes `_stimulate`, which itself unsets the currently selected stimulus at the end of the stimulation. You have to invoke `select_stimulus` again before you can call `stimulate` again.

stimuli

A list of all stimuli added to this stimulation apparatus.

See also:

`add_stimulus`, `remove_stimulus`

stimulus

The currently selected stimulus.

This stimulus will be presented when invoking `stimulate`. It can be set using `select_stimulus`.

See also:

`select_stimulus`, `stimulate`

4.2 AnalogInput

class pphelper.hardware.**AnalogInput**(*ni_input_line=u'Dev1/ai0'*, *ni_trigger_line=None*, *sampling_duration=3*, *sampling_rate=2000*, *ni_task_name=u'AnalogInput'*)

Analog data acquisition using a National Instruments board.

get_data()

Return the acquired data.

If the acquisition is not finished by the time this method is called, it will first wait until finished (blocking other processing) and then return the data.

samples_to_acquire

The number of samples to acquire in the acquisition. This is the product of the sampling rate and the sampling duration.

sampling_duration
The duration (in seconds) of data acquisition.

sampling_rate
The sampling rate (in Hz) of the analog data acquisition.

trigger()
Return the acquired data.

If the acquisition is not finished by the time this method is called, it will first wait until finished (blocking other processing) and then return the data.

4.3 Gustometer

class pphelper.hardware.**Gustometer**(*pulse_duration=0.1*, *pause_duration=0.2*, *gusto_ip=u'192.168.0.1'*, *gusto_port=40175*, *local_ip=u'192.168.0.10'*, *local_port=40176*, *ni_trigger_out_line=u'Dev1/PFI1'*, *ni_trigger_in_line=u'Dev1/ctr0'*, *ni_trigger_out_task_name=u'GustometerOut'*, *ni_trigger_in_task_name=u'GustometerIn'*, *use_threads=True*)

Provide an interface to the Burghart GU002 gustometer.

add_stimulus(*name*, *classnum*, *trigger_time=None*, *replace=False*, ***kwargs*)
Add a stimulus to the stimulus set of this apparatus.

Parameters

- **name** (*string*) – A unique identifier of the stimulus to add.
- **classnum** (*int*) – The stimulus class number, as defined in the Gusto Control software.
- **trigger_time** (*float, optional*) – The time (in terms of the `psychopy.core.getTime` timebase) at which the stimulation should be triggered. If `None`, trigger immediately. Defaults to `None`.
- **replace** (*bool, optional*) – Whether an already existing stimulus of the same name should be replaced or not. Defaults to `False`.

Notes

Any additional keyword arguments will be added as additional stimulus properties.

See also:

`select_stimulus()`, `remove_stimulus()`, `stimulate()`

load_classfile(*filename*)
Load a classes file in the Gusto Control software.

Parameters **filename** (*string*) – The filename of the classes file to be loaded, including the file extension (typically *.cla*).

remove_stimulus(*name*)
Remove the specified stimulus, identified by its unique name, from the stimulus list.

Parameters **name** (*string*) – The unique name of the stimulus to remove.

select_stimulus (*name*)

Select the specified stimulus for the next stimulation.

Parameters **name** (*string*) – The unique name of the stimulus to select.

stimulate (*blocking_wait=False*)

Start the stimulation with the currently selected stimulus.

The trigger pulse will open the valves. They are then left open for the intended duration of the stimulus. After that, they will be switched to the offset state specified by the stimulus.

Parameters **blocking_wait** (*bool, optional*) – Specifies whether the stimulation thread should be *joined* or not, i.e. whether we should wait for it to finish (blocking other operations), or return immediately. This parameter will be ignored if threads are not used for stimulation. Defaults to *False*, i.e. non-blocking behavior.

See also:

`add_stimulus()`, `select_stimulus()`

Notes

`stimulate` invokes `_stimulate`, which itself unsets the currently selected stimulus at the end of the stimulation. You have to invoke `select_stimulus` again before you can call `stimulate` again.

stimuli

A list of all stimuli added to this stimulation apparatus.

See also:

`add_stimulus`, `remove_stimulus`

stimulus

The currently selected stimulus.

This stimulus will be presented when invoking `stimulate`. It can be set using `select_stimulus`.

See also:

`select_stimulus`, `stimulate`

trigger_conf (*duration=0.9*, *int_taste=100*, *int_bg=100*)

Configure the trigger on the gusto.

Parameters

- **duration** (*int, optional*) – Trigger duration (stimulation duration) in seconds. Defaults to 0.9s.
- **int_taste** (*int*) – Taste intensity in percent. Defaults to 100%.
- **int_bg** (*int*) – Background intensity in percent. Defaults to 100%.

CHAPTER 5

The `image` Module

5.1 pphelper.image

5.1.1 Provides

- `fft_image` : Perform an FFT on the supplied image array.
- **`lowpass_filter_image`** [Load an image from a file, and low-pass] filter via a Gaussian kernel.

`pphelper.image.`**`fft_image`**(*image*)

Perform an FFT on the supplied image array.

Parameters **image** (*ndarray*) – An array of the image

Returns A namedtuple containing the the fast-fourier transform, the amplitude and phase.

Return type namedtuple

`pphelper.image.`**`lowpass_filter_image`**(*image=None*, *filename=None*, *flatten=False*, *sigma=3*)

Load an image from a file, and low-pass filter via a Gaussian kernel.

Parameters

- **image** (*ndarray, optional*) – The image to be processed. This will usually have been created using *scipy.misc.imread* or a similar function. If this argument is present, *filename* will be ignored.
- **filename** (*string, optional*) – The name of the image file to load and process. Will be ignored is *image_array* is not *None*.
- **flatten** (*bool, optional*) – Whether to "flatten" the image before filtering, i.e. convert it to grayscale.
- **sigma** (*scalar, optional*) – The standard deviation for Gaussian kernel. See *scipy.ndimage.filters.gaussian_filter*.

Returns The lowpass-filtered image.

Return type ndarray

See also:

`scipy.ndimage.gaussian_filter()`

CHAPTER 6

The `utils` Module

Some functions that come in handy when working with psychophysics datasets.

6.1 Provides

- `add_zero_padding` : Convert numbers (typically participant IDs) to strings of specific length, with leading zeros where necessary.
- `get_max_from_list` : Return the maximum value from a list or a list of lists.

`pphelper.utils.`**`add_zero_padding`**(*data*, *length=3*, *return_series=True*)

Convert input values to strings and add a zero-padding.

Parameters

- **data** (*array_like*) – The data to process.
- **length** (*int, optional*) – The desired length of the padded output values.
- **return_series** (*bool, optional*) – If *True*, return a pandas *Series* object. If false, return a numpy array.

Returns **result** – The padded input vector. All strings are created as unicode literals.

Return type Series or ndarray

Notes

If `length` is less than the length of one of the elements, these elements will be converted to strings only and returned. They will obviously not be zero-padded, but they will also not be truncated.

`pphelper.utils.`**`get_max_from_list`**(*x*)

Return the maximum value from a list or a list of lists.

Parameters **x** (*list*) – A list or a list of lists.

Returns The maximum value.

Return type float

Python Module Index

p

pphelper, 1
pphelper.image, 17
pphelper.racemodel, 3
pphelper.sdt, 11
pphelper.utils, 19

A

a_prime() (in module pphelper.sdt), 11
add_stimulus() (pphelper.hardware.Gustometer method), 15
add_stimulus() (pphelper.hardware.Olfactometer method), 13
add_zero_padding() (in module pphelper.utils), 19
AnalogInput (class in pphelper.hardware), 14

C

criterion() (in module pphelper.sdt), 11

D

d_prime() (in module pphelper.sdt), 12

F

fft_image() (in module pphelper.image), 17

G

gen_cdf() (in module pphelper.racemodel), 3
gen_cdfs_from_dataframe() (in module pphelper.racemodel), 4
gen_cdfs_from_list() (in module pphelper.racemodel), 6
gen_percentiles() (in module pphelper.racemodel), 8
gen_step_fun() (in module pphelper.racemodel), 8
get_data() (pphelper.hardware.AnalogInput method), 14
get_max_from_list() (in module pphelper.utils), 19
get_percentiles_from_cdf() (in module pphelper.racemodel), 9
Gustometer (class in pphelper.hardware), 15

L

load_classfile() (pphelper.hardware.Gustometer method), 15
lowpass_filter_image() (in module pphelper.image), 17

O

Olfactometer (class in pphelper.hardware), 13

P

pphelper (module), 1
pphelper.image (module), 17
pphelper.racemodel (module), 3
pphelper.sdt (module), 11
pphelper.utils (module), 19

R

remove_stimulus() (pphelper.hardware.Gustometer method), 15
remove_stimulus() (pphelper.hardware.Olfactometer method), 13

S

samples_to_acquire (pphelper.hardware.AnalogInput attribute), 14
sampling_duration (pphelper.hardware.AnalogInput attribute), 14
sampling_rate (pphelper.hardware.AnalogInput attribute), 15
select_stimulus() (pphelper.hardware.Gustometer method), 15
select_stimulus() (pphelper.hardware.Olfactometer method), 14
stimulate() (pphelper.hardware.Gustometer method), 16
stimulate() (pphelper.hardware.Olfactometer method), 14
stimuli (pphelper.hardware.Gustometer attribute), 16
stimuli (pphelper.hardware.Olfactometer attribute), 14
stimulus (pphelper.hardware.Gustometer attribute), 16
stimulus (pphelper.hardware.Olfactometer attribute), 14
sum_cdfs() (in module pphelper.racemodel), 9

T

trigger() (pphelper.hardware.AnalogInput method), 15
trigger_conf() (pphelper.hardware.Gustometer method), 16