pyLGN Documentation

Milad H. Mobarhan

Nov 30, 2018

Contents

pyLGN is a visual stimulus-driven simulator of spatiotemporal cell responses in the early part of the visual system consisting of the retina, lateral geniculate nucleus (LGN) and primary visual cortex. The simulator is based on a mechanistic, firing rate model that incorporates the influence of thalamocortical loops, in addition to the feedforward responses. The advantage of the simulator lies in its computational and conceptual ease, allowing for fast and comprehensive exploration of various scenarios for the organization of the LGN circuit.

CHAPTER 1

Installation

1.1 Pre-configured installation (recommended)

It's strongly recommended that you use Anaconda to install pyLGN along with its compiled dependencies.

With Anaconda or Miniconda:

```
conda install -c defaults -c conda-forge -c cinpla pylgn
```

pyLGN can also be installed by cloning the Github repository:

```
$ git clone https://github.com/miladh/pylgn
$ cd /path/to/pylgn
$ python setup.py install
```

Note that dependencies must be installed independently.

CHAPTER 2

Getting Started

2.1 Install

With Anaconda or Miniconda:

```
conda install -c defaults -c conda-forge -c cinpla pylgn
```

2.2 Minimal example

This example shows a minimal network consisting of a ganglion cell population and a relay cell population. A space-time separable impulse-response function is assumed for the ganglion cell, with a spatial part modeled as a difference of Gaussian (DoG) function while the temporal part is a delta function. The connectivity kernel between ganglion cells and relay cells is also assumed to be space-time separable, with a spatial part modeled as a Gaussian function while the temporal part is a delta function. The stimulus is full-field grating.

The complete code and a step-by-step explanation is given below:

```
import pylgn
import pylgn.kernels.spatial as spl
import pylgn.kernels.temporal as tpl

# create network
network = pylgn.Network()

# create integrator
integrator = network.create_integrator(nt=5, nr=7, dt=1, dr=1)

# create neurons
ganglion = network.create_ganglion_cell()
relay = network.create_relay_cell()
```

(continues on next page)

(continued from previous page)

```
# create kernels
Krg_r = spl.create_gauss_ft()
Krg_t = tpl.create_delta_ft()

# connect neurons
network.connect(ganglion, relay, (Krg_r, Krg_t))

# create stimulus
k_g = integrator.spatial_angular_freqs[3]
w_g = -integrator.temporal_angular_freqs[1]
stimulus = pylgn.stimulus.create_fullfield_grating_ft(angular_freq=w_g,
                                                      wavenumber=k_g,
                                                      orient=0.0)
network.set_stimulus(stimulus)

# compute
network.compute_response(relay)

# visulize
pylgn.plot.animate_cube(relay.response, title="Relay cell response")
```

2.2.1 Create network

First step is to import pyLGN, including the spatial and temporal kernels, and create a network:

```
import pylgn
import pylgn.kernels.spatial as spl
import pylgn.kernels.temporal as tpl

network = pylgn.Network()
```

2.2.2 Create integrator

Next we create an integrator with 2^{nt} and 2^{ns} spatial and temporal points, respectively. The temporal and spatial resolutions are `dt=1` (ms) and `dr=0.1` (deg), respectively. Note that if units are not given for the resolutions, "ms" and "deg" are used by default.

```
integrator = network.create_integrator(nt=5, nr=7, dt=1, dr=0.1)
```

2.2.3 Create neurons

Cells can be added to the network using `create_<name>_cell()` method:

```
ganglion = network.create_ganglion_cell()
relay = network.create_relay_cell()
```

Note: The impulse-response function of ganglion cells can be set in two ways:

- It can either be given as an argument `kernel` when the neuron object is created using *`create_ganglion_cell()`*. If no argument is given, a spatial DoG function and a temporal biphasic function is used.

- The second option is to use the *set_kernel()* method after that the neuron object is created.

The various neuron attributes are stored in a dictionary on the neuron objects:

```
>>> print(ganglion.annotations)
{'background_response': array(0.0) * 1/s, 'kernel': {'spatial': {'center': {
↪'params': {'A': 1, 'a': array(0.62) * deg}, 'type': 'create_gauss_ft'},
↪'surround': {'params': {'A': 0.85, 'a': array(1.26) * deg}, 'type':
↪'create_gauss_ft'}, 'type': 'create_dog_ft'}, 'temporal': {'params': {
↪'delay': array(0.0) * ms}, 'type': 'create_delta_ft'}}}
```

2.2.4 Connect neurons

We use a separable kernel between the ganglion cells and relay cells. The *connect()* method has the following signature: `connect(source, target, kernel, weight)`, where source and target are the source and target neurons, respectively, kernel is the connectivity kernel, and weight is the connection weight (default is 1). If a separable kernel is used a tuple consisting of the spatial and temporal part is given as kernel. The order of kernels in the tuple does not matter.

```
Krg_r = spl.create_gauss_ft()
Krg_t = tpl.create_delta_ft()

network.connect(ganglion, relay, (Krg_r, Krg_t))
```

Note: The kernel parameters can be received using:

```
>>> print(pylgn.closure_params(Krg_r))
{'params': {'A': 1, 'a': array(0.62) * deg}, 'type': 'create_gauss_ft'}
```

2.2.5 Create stimulus

A full-field grating stimulus has several parameters including `angular_freq`, `wavenumber`, and `orient`. If you want to use the analytical expression for the Fourier transform of the grating stimulus, you have to make sure that the chosen frequencies exist in the `integrator.spatial_angular_freqs` and `integrator.temporal_angular_freqs` determined by the number of points and resolutions. In this example we use frequencies from these arrays:

```
k_g = integrator.spatial_angular_freqs[3]
w_g = integrator.temporal_angular_freqs[1]
stimulus = pylgn.stimulus.create_fullfield_grating_ft(angular_freq=w_g,
                                                      wavenumber=k_g,
                                                      orient=0.0)
network.set_stimulus(stimulus)
```

Note: If you wish to use frequencies that does not exist in the grid, numerical integration can be used. In such cases the inverse Fourier transform of the stimulus must be given. Then `network.set_stimulus(stimulus, compute_fft=True)` method can be used to set the stimulus.

2.2.6 Compute response

The lines below computes the response of the relay cells and animate their activity over time:

```
network.compute_response(relay)
pylgn.plot.animate_cube(relay.response)
```

CHAPTER 3

Examples

3.1 From papers

3.1.1 Spatial receptive-field of nonlagged cells in dLGN of cat

In this example figure 5 in Einevoll et al. (2000) is reproduced. In the code below the size tuning curve for the ganglion cell and the relay cell is calculated. The resulting figure shown below:

```
import numpy as np
import quantities as pq
import matplotlib.pyplot as plt

import pylgn
import pylgn.kernels as kernel


patch_diameter = np.linspace(0, 14, 50) * pq.deg
R_g = np.zeros(len(patch_diameter)) / pq.s
R_r = np.zeros(len(patch_diameter)) / pq.s

# create network
network = pylgn.Network()

# create integrator
integrator = network.create_integrator(nt=1, nr=8, dt=1*pq.ms, dr=0.1*pq.deg)

# create neurons
ganglion = network.create_ganglion_cell(background_response=36.8/pq.s)
relay = network.create_relay_cell(background_response=9.1/pq.s)

# create kernels
Wg_r = kernel.spatial.create_dog_ft(A=-1, a=0.62*pq.deg, B=-0.85, b=1.26*pq.deg)
Krig_r = kernel.spatial.create_gauss_ft(A=1, a=0.88*pq.deg)
```

(continues on next page)

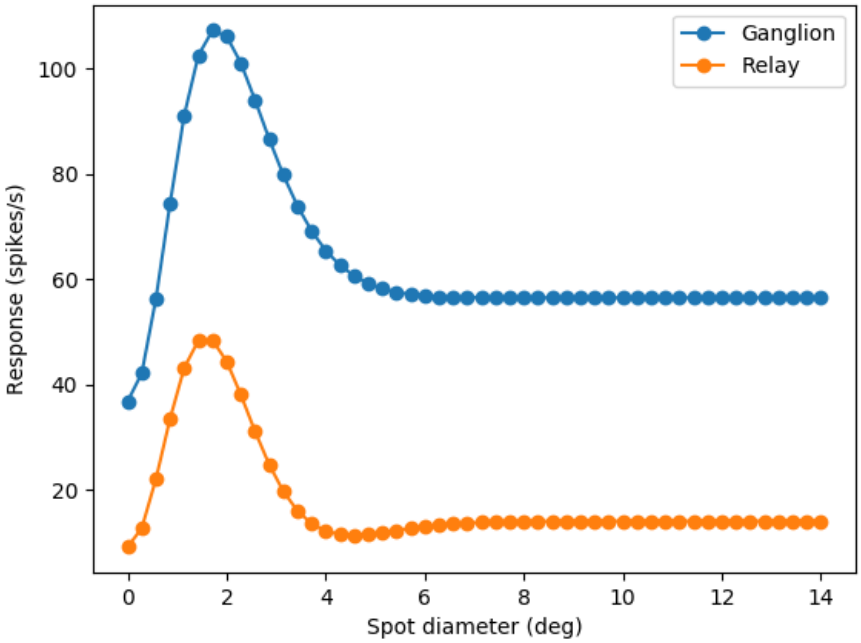


Fig. 1: Area summation curve for an OFF-center ganglion cell and relay cell with dark spot stimulus.

(continued from previous page)

```
Krg_r = kernel.spatial.create_delta_ft()

# connect neurons
ganglion.set_kernel((Wg_r, kernel.temporal.create_delta_ft()))
network.connect(ganglion, relay, (Krg_r, kernel.temporal.create_delta_ft()), weight=0.
↪81)
network.connect(ganglion, relay, (Krig_r, kernel.temporal.create_delta_ft()), weight=-
↪0.56)

for i, d in enumerate(patch_diameter):
    # create stimulus
    stimulus = pylgn.stimulus.create_patch_grating_ft(patch_diameter=d, contrast=-131.
↪3)
    network.set_stimulus(stimulus)

    # compute
    network.compute_response(ganglion, recompute_ft=True)
    network.compute_response(relay, recompute_ft=True)

    R_g[i] = ganglion.center_response[0]
    R_r[i] = relay.center_response[0]

# visualize
plt.plot(patch_diameter, R_g, '-o', label="Ganglion")
plt.plot(patch_diameter, R_r, '-o', label="Relay")
plt.xlabel("Spot diameter (deg)")
plt.ylabel("Response (spikes/s)")
```

(continues on next page)

(continued from previous page)

```
plt.legend()
plt.show()
```

3.1.2 Response of the DoG model to patch grating

In this example script figure 4 in Einevoll et al. (2005) is reproduced. In the code below the response of the difference of Gaussians (DoG) model to circular drifting-grating patches is calculated:

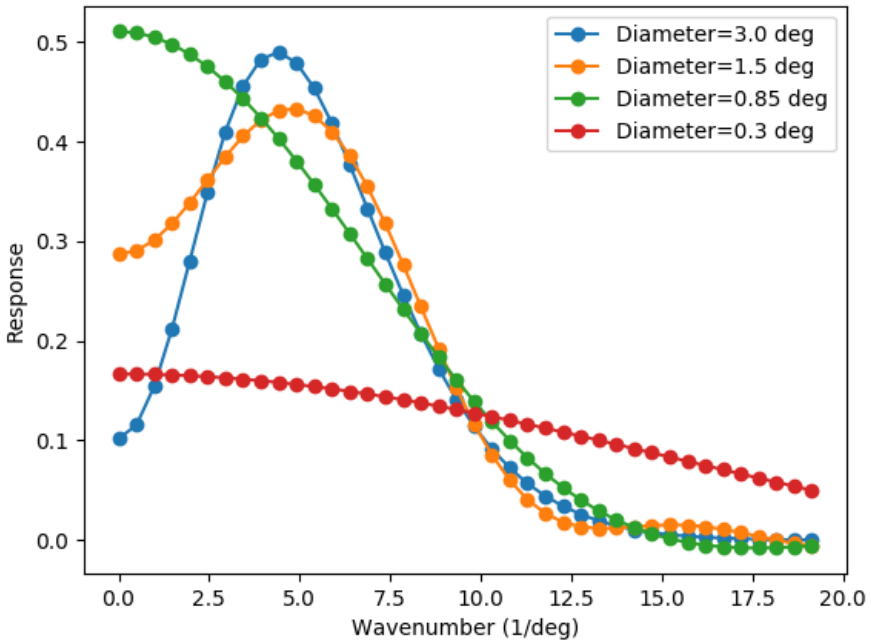


Fig. 2: Spatial frequency tuning curve of patch-grating responses for four different patch diameters for the DoG model.

```
import quantities as pq
import numpy as np
import matplotlib.pyplot as plt

import pylgn
import pylgn.kernels as kernel

k_max_id = 40
patch_diameter = np.array([3, 1.5, 0.85, 0.3]) * pq.deg
response = np.zeros([k_max_id, len(patch_diameter)]) / pq.s

# create network
network = pylgn.Network()

# create integrator
integrator = network.create_integrator(nt=1, nr=7, dt=1*pq.ms, dr=0.1*pq.deg)
spatial_angular_freqs = integrator.spatial_angular_freqs[:k_max_id]
```

(continues on next page)

(continued from previous page)

```
# create kernels
Wg_t = kernel.temporal.create_delta_ft()
Wg_r = kernel.spatial.create_dog_ft(A=1, a=0.3*pq.deg, B=0.9, b=0.6*pq.deg)

# create neuron
ganglion = network.create_ganglion_cell(kernel=(Wg_r, Wg_t))

for j, d in enumerate(patch_diameter):
    for i, k_d in enumerate(spatial_angular_freqs):
        # create stimulus
        stimulus = pylgn.stimulus.create_patch_grating_ft(wavenumber=k_d, patch_
↪diameter=d)
        network.set_stimulus(stimulus)

        # compute
        network.compute_response(ganglion, recompute_ft=True)
        response[i, j] = ganglion.center_response[0]

# visualize
for d, R in zip(patch_diameter, response.T):
    plt.plot(spatial_angular_freqs, R, '-o', label="Diameter={}".format(d))

plt.xlabel("Wavenumber (1/deg)")
plt.ylabel("Response")
plt.legend()
plt.show()
```

3.1.3 Extended DoG model with cortical feedback

In this example figure 6 in Einevoll et al. (2012) is reproduced, where the response of the extended difference-of-Gaussians (eDoG) model to flashing-spot is calculated. The resulting figure is shown below:

```
import quantities as pq
import numpy as np
import matplotlib.pyplot as plt

import pylgn
import pylgn.kernels as kernel

# fb weights:
fb_weights = [0, -1.5]

# diameters
patch_diameter = np.linspace(0, 6, 50) * pq.deg
response = np.zeros([len(patch_diameter), len(fb_weights)]) / pq.s

for j, w_c in enumerate(fb_weights):
    # create network
    network = pylgn.Network()

    # create integrator
    integrator = network.create_integrator(nt=1, nr=7, dt=1*pq.ms, dr=0.1*pq.deg)

    # create kernels
    delta_t = kernel.temporal.create_delta_ft()
```

(continues on next page)

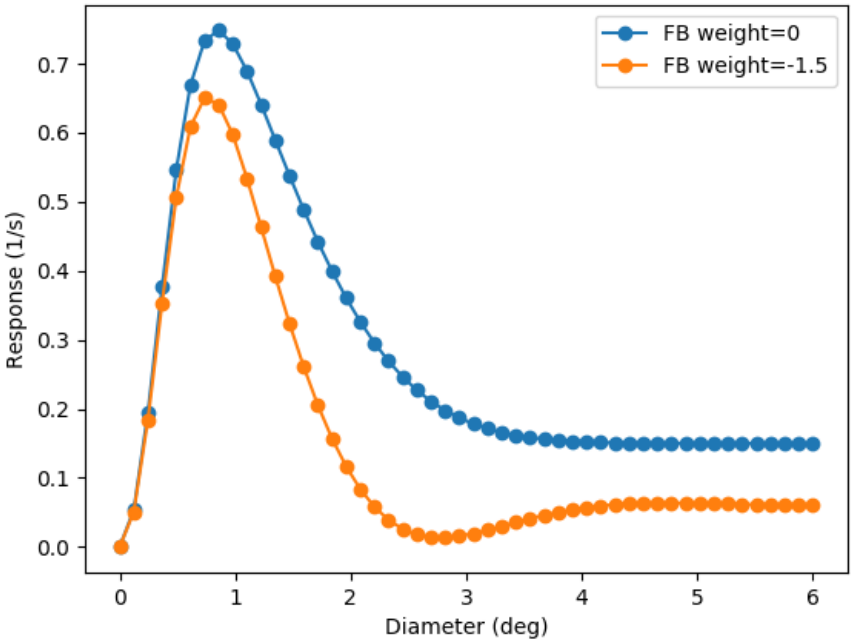


Fig. 3: Flashing-spot response as a function of spot diameter for different feedback weights.

(continued from previous page)

```
    delta_s = kernel.spatial.create_delta_ft()
    Wg_r = kernel.spatial.create_dog_ft(A=1, a=0.25*pq.deg, B=0.85, b=0.83*pq.deg)
    Krc_r = kernel.spatial.create_gauss_ft(A=1, a=0.83*pq.deg)

    # create neurons
    ganglion = network.create_ganglion_cell(kernel=(Wg_r, delta_t))
    relay = network.create_relay_cell()
    cortical = network.create_cortical_cell()

    # connect neurons
    network.connect(ganglion, relay, (delta_s, delta_t), 1.0)
    network.connect(cortical, relay, (Krc_r, delta_t), w_c)
    network.connect(relay, cortical, (delta_s, delta_t), 1.0)

    for i, d in enumerate(patch_diameter):
        # create stimulus
        stimulus = pylgn.stimulus.create_patch_grating_ft(wavenumber=0,
                                                          patch_diameter=d)
        network.set_stimulus(stimulus)

        # compute
        network.compute_response(relay, recompute_ft=True)

        response[i, j] = relay.center_response[0]

    # clear network
    network.clear()
```

(continues on next page)

(continued from previous page)

```
# visualize
plt.plot(patch_diameter, response[:, 0], '-o', label="FB weight={}".format(fb_
↪weights[0]))
plt.plot(patch_diameter, response[:, 1], '-o', label="FB weight={}".format(fb_
↪weights[1]))
plt.xlabel("Diameter (deg)")
plt.ylabel("Response (1/s)")
plt.legend()
plt.show()
```

The response to patch-grating can be calculated using the same code with a small modification in stimulus: `wavenumber=integrator.spatial_angular_freqs[4]` which corresponds to wavenumber ~2.0/deg.

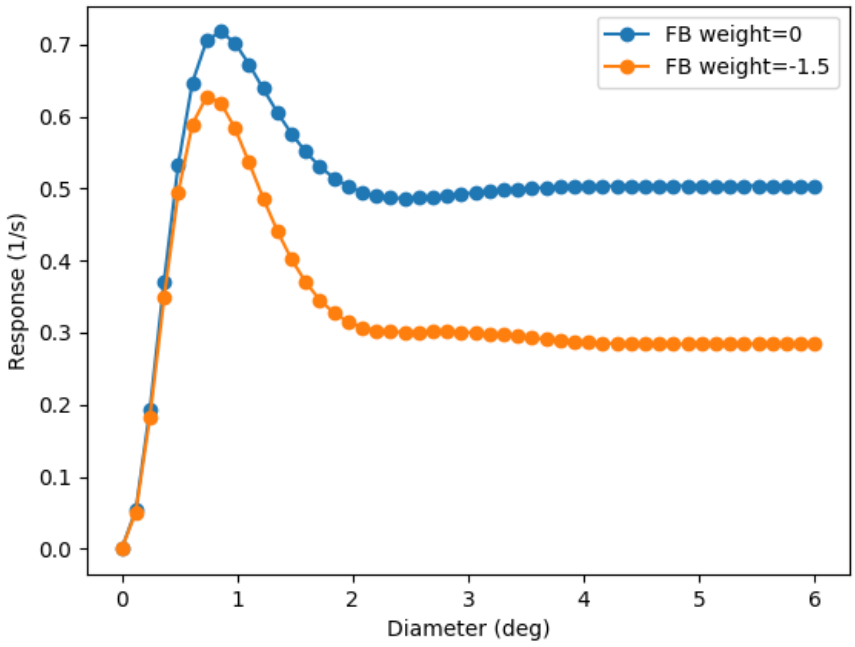


Fig. 4: Patch-grating response (wavenumber ~2.0/deg) as a function of spot diameter for different feedback weights.

3.2 Natural scenes

3.2.1 Using natural scenes and natural movies as stimulus

Natural scene

In this example a static image is shown in 80 ms after a 40 ms delay.

```
import pylgn
import pylgn.kernels.spatial as spl
```

(continues on next page)

(continued from previous page)

```
import pylgn.kernels.temporal as tpl
import quantities as pq

# create network
network = pylgn.Network()

# create integrator
integrator = network.create_integrator(nt=8, nr=9, dt=1*pq.ms, dr=0.1*pq.deg)

# create kernels
Wg_r = spl.create_dog_ft()
Wg_t = tpl.create_biphasic_ft()

# create neurons
ganglion = network.create_ganglion_cell(kernel=(Wg_r, Wg_t))

# create stimulus
stimulus = pylgn.stimulus.create_natural_image(filenames="natural_scene.png",
                                               delay=40*pq.ms,
                                               duration=80*pq.ms)
network.set_stimulus(stimulus, compute_fft=True)

# compute
network.compute_response(ganglion)

# visulize
pylgn.plot.animate_cube(ganglion.response,
                        title="Ganglion cell responses",
                        dt=integrator.dt.rescale("ms"))
```

The response of the ganglion cells is shown as a heatmap from blue to red (low to high response).

Natural movie

Natural movies can be given as GIFs:

```
stimulus = pylgn.stimulus.create_natural_movie(filenames="natural_scene.gif")
```

Note: If GIF file do not have a "duration" key (time between frames) 30 ms is used by default. See Pillow documentation for details.

3.2.2 Generate spike trains

In this example a static image is shown in 80 ms after a 40 ms delay.

```
import pylgn
import pylgn.kernels.spatial as spl
import pylgn.kernels.temporal as tpl
import quantities as pq
```

(continues on next page)

(continued from previous page)

```
# create network
network = pylgn.Network()

# create integrator
integrator = network.create_integrator(nt=7, nr=7, dt=2*pq.ms, dr=0.4*pq.deg)

# create kernels
Wg_r = spl.create_dog_ft()
Wg_t = tpl.create_biphasic_ft()

# create neurons
ganglion = network.create_ganglion_cell(kernel=(Wg_r, Wg_t))

# create stimulus
stimulus = pylgn.stimulus.create_natural_image(filenames="natural_scene.png",
                                               delay=40*pq.ms,
                                               duration=80*pq.ms)
network.set_stimulus(stimulus, compute_fft=True)

# compute
network.compute_response(ganglion)
```

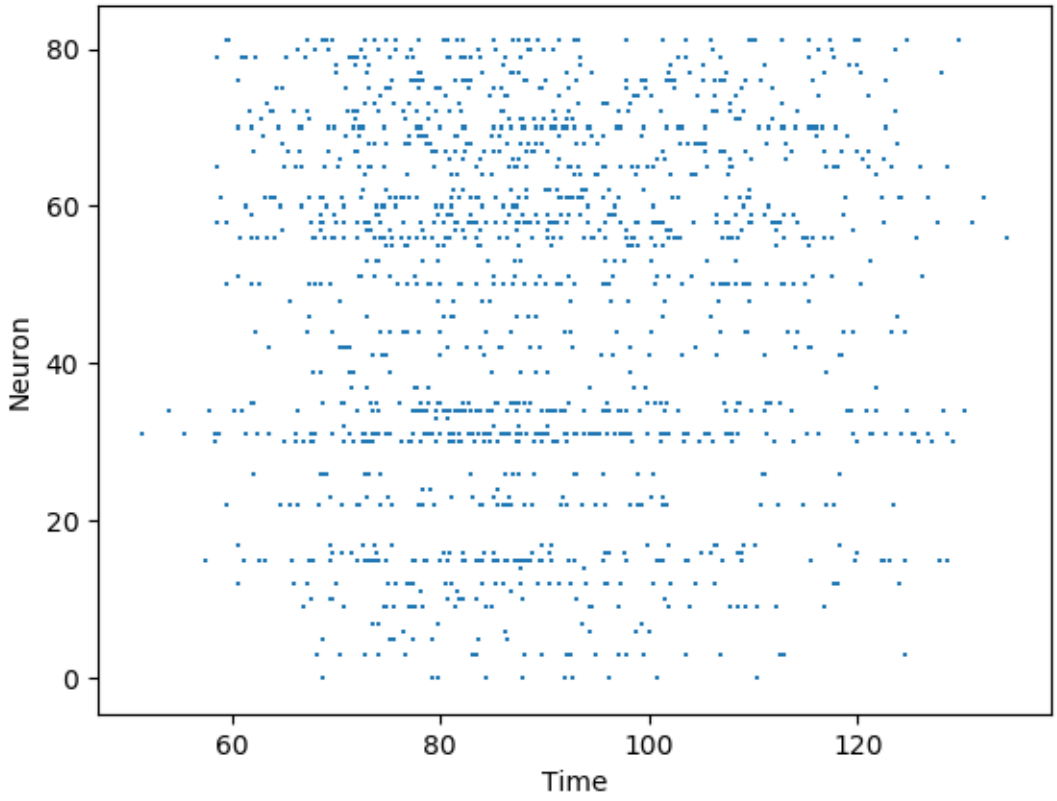
The calculated rates can be converted to spikes via a nonstationary Poisson process:

```
import pylgn.tools as tls

# apply static nonlinearity and scale rates
rates = ganglion.response
rates = tls.heaviside_nonlinearity(rates)
rates = tls.scale_rates(rates, 60*pq.Hz)

# generate spike trains
spike_trains = tls.generate_spike_train(rates, integrator.times)

# visulize
pylgn.plot.animate_spike_activity(spike_trains,
                                  times=integrator.times,
                                  positions=integrator.positions,
                                  title="Spike activity")
```

In the animation below the generated spikes at each location are shown as dots:

A simple raster plot of individual locations can be created using:

```
pylgn.plot.raster_plot(spike_trains.flatten()[::200])
```


80
60
40
20
0
Neuron
60
80
100
120
Time

CHAPTER 4

Network

class pylgn.core.**Network**(*memory_efficient=False*)

Network class

Variables

- **neurons** (*list*) – List with pylgn.Neuron objects
- **integrator** (*pylgn.Integrator*) – Integrator object
- ***stimulus*** (*pylgn.Stimulus*) – Stimulus object

__init__(*memory_efficient=False*)

Network constructor

clear()

Clears the neuron list.

compute_irf(*neuron*, *recompute_ft=False*)

Computes the impulse-response function of a neuron.

Parameters

- **neuron** (*pylgn.Neuron*)
- **recompute_ft** (*bool*) – If True the Fourier transform is recalculated.

compute_irf_ft(*neuron*)

Computes the Fourier transform of the impulse-response function of a neuron.

Parameters **neuron** (*pylgn.Neuron*)

compute_response(*neuron*, *recompute_ft=False*)

Computes the response of a neuron.

Parameters

- **neuron** (*pylgn.Neuron*)
- **recompute_ft** (*bool*) – If True the Fourier transform is recalculated.

compute_response_ft (*neuron*, *recompute_irf_ft=False*)
Computes the Fourier transform of the response of a neuron.

Parameters **neuron** (*pylgn.Neuron*)

connect (*source*, *target*, *kernel*, *weight=1.0*)
Connect neurons.

Parameters

- **source** (*pylgn.Neuron*) – Source neuron
- **target** (*pylgn.Neuron*) – Target neuron
- **kernel** (*function*) – Connectivity kernel
- **weight** (*float*) – Connectivity weight

create_cortical_cell (*background_response=array(0.) * 1/s*, *annotations={}*)
Create cortical cell

Parameters

- **background_response** (*quantity scalar*) – Background activity.
- **annotations** (*dict*) – Dictionary with various annotations.

Returns **out** – Cortical object

Return type *pylgn.Cortical*

create_descriptive_neuron (*background_response=array(0.) * 1/s*, *kernel=None*, *annotations={}*)
Create descriptive neuron

Parameters

- **background_response** (*quantity scalar*) – Background activity.
- **kernel** (*function*) – Impulse-response function.
- **annotations** (*dict*) – Dictionary with various annotations.

Returns **out** – Descriptive neuron object

Return type pylgn.DescriptiveNeuron

create_ganglion_cell (*background_response=array(0.) * 1/s*, *kernel=None*, *annotations={}*)
Create ganglion cell

Parameters

- **background_response** (*quantity scalar*) – Background activity.
- **kernel** (*function*) – Impulse-response function.
- **annotations** (*dict*) – Dictionary with various annotations.

Returns **out** – Ganglion object

Return type pylgn.Ganglion

create_integrator (*nt*, *nr*, *dt*, *dr*)
Create and set integrator

Parameters

- **nt** (*int*) – The power to raise 2 to. Number of temporal points is 2**nt.
- **nr** (*int*) – The power to raise 2 to. Number of spatial points is 2**nr.

- **dt** (*quantity scalar*) – Temporal resolution
- **dr** (*quantity scalar*) – Spatial resolution

Returns **out** – Integrator object

Return type pylgn.Integrator

create_relay_cell (*background_response=array(0.) * 1/s*, *annotations={}*)
Create relay cell

Parameters

- **background_response** (*quantity scalar*) – Background activity.
- **annotations** (*dict*) – Dictionary with various annotations.

Returns **out** – Relay object

Return type pylgn.Relay

set_stimulus (*closure*, *compute_fft=False*)
Sets stimulus.

Parameters

- **closure** (*callable (closure)*) – stimulus function. If compute_fft is False the stimulus function should be the Fourier transform of the stimulus.
- **compute_fft** (*bool*) – If True numerical integration is used to calculate the Fourier transform of the stimulus.

pylgn.core.**closure_params** (*closure*)
Stores closure parameters in a dict

Parameters **closure** (*function*) – A closure function

Returns **out** – Dictionary

Return type dict

CHAPTER 5

Neurons

Documentation of Neuron classes.

5.1 Neuron

class pylgn.core.**Neuron**(*background_response*, *annotations*)

Neuron base class.

Variables

- ***center_response*** –
- **background_response** (*quantity scalar*) – Background activity.
- **annotations** (*dict*) – Dictionary with various annotations on the Neuron object.
- **connections** (*dict*) – Dictionary with connected neurons including the connectivity kernel and weight.
- **response** (*quantity array*) – Spatiotemporal response
- **response_ft** (*quantity array*) – Fourier transformed response
- **irf** (*quantity array*) – Spatiotemporal impulse-response function
- **irf_ft** (*quantity array*) – Fourier transformed impulse-response function

__init__(*background_response*, *annotations*)

Neuron constructor

Parameters

- **background_response** (*quantity scalar*) – Background activity.
- **annotations** (*dict*) – Dictionary with various annotations on the Neuron object.

add_connection(*neuron*, *kernel*, *weight*)

Add connection to another neuron.

Parameters

- **neuron** (*pylgn.Neuron*) – Source neuron
- **kernel** (*functions*) – Connectivity kernel
- **weight** (*float*) – Connectivity weight

annotate(*annotations*)
Add annotations to a Neuron object.

Parameters **annotations** (*dict*) – Dictionary containing annotations

center_response
Response of neuron in the center of grid over time

Returns **out** – Response of neuron in the center of grid over time

Return type quantity array

evaluate_irf_ft(*w*, *kx*, *ky*)
Evaluates the Fourier transform of impulse-response function

5.2 Ganglion class

class pylgn.core.**Ganglion**(*background_response*, *kernel*, *annotations={}*)

__init__(*background_response*, *kernel*, *annotations={}*)
Ganglion constructor

Parameters

- **background_response** (*quantity scalar*) – Background activity.
- **kernel** (*function*) – Impulse-response function.
- **annotations** (*dict*) – Dictionary with various annotations.

evaluate_irf_ft(*w*, *kx*, *ky*)
Evaluates the Fourier transform of impulse-response function

set_kernel(*kernel*)
Set the impulse-response function.

Parameters **kernel** (*func or tuple*) – Fourier transformed kernel/ tuple of Fourier transformed spatial and temporal kernel

5.3 Relay class

class pylgn.core.**Relay**(*background_response*, *annotations={}*)

__init__(*background_response*, *annotations={}*)
Relay constructor

Parameters

- **background_response** (*quantity scalar*) – Background activity.
- **annotations** (*dict*) – Dictionary with various annotations.

evaluate_irf_ft(*w*, *kx*, *ky*)
Evaluates the Fourier transform of impulse-response function

5.4 Cortical class

class pylgn.**Cortical**(*background_response*, *annotations={}*)

__init__(*background_response*, *annotations={}*)
Cortical constructor

Parameters

- **background_response** (*quantity scalar*) – Background activity.
- **annotations** (*dict*) – Dictionary with various annotations.

evaluate_irf_ft(*w*, *kx*, *ky*)
Evaluates the Fourier transform of impulse-response function

5.5 DescriptiveNeuron class

class pylgn.core.**DescriptiveNeuron**(*background_response*, *kernel*, *annotations={}*)

__init__(*background_response*, *kernel*, *annotations={}*)
Descriptive neuron constructor

Parameters

- **background_response** (*quantity scalar*) – Background activity.
- **kernel** (*function*) – Impulse-response function.
- **annotations** (*dict*) – Dictionary with various annotations.

evaluate_irf_ft(*w*, *kx*, *ky*)
Evaluates the Fourier transform of impulse-response function

set_kernel(*kernel*)
Set the impulse-response function.

Parameters **kernel** (*func or tuple*) – Fourier transformed kernel/ tuple of Fourier transformed spatial and temporal kernel

CHAPTER 6

Integrator

class pylgn.core.**Integrator**(*nt*, *nr*, *dt*, *dr*)

Integrator class for fast Fourier transform calculations.

Variables

- ***times*** –
- ***positions*** –
- ***temporal_angular_freqs*** –
- ***spatial_angular_freqs*** –
- **Nt** (*int*) – Number of spatial points.
- **Nr** (*int*) – Number of temporal points
- **dt** (*quantity scalar*) – Temporal resolution
- **dr** (*quantity scalar*) – Spatial resolution
- **dw** (*quantity scalar*) – Temporal frequency resolution
- **dk** (*quantity scalar*) – Spatial frequency resolution

__init__(*nt*, *nr*, *dt*, *dr*)

Integrator constructor

Parameters

- **nt** (*int*) – The power to raise 2 to. Number of temporal points is 2**nt.
- **nr** (*int*) – The power to raise 2 to. Number of spatial points is 2**nr.
- **dt** (*quantity scalar*) – Temporal resolution
- **dr** (*quantity scalar*) – Spatial resolution

compute_fft(*cube*)

Computes fast Fourier transform.

Parameters **cube** (*array_like*) – input array (3-dimensional)

Returns **out** – transformed array

Return type array_like

compute_inverse_fft(*cube*)
Computes inverse fast Fourier transform.

Parameters **cube** (*array_like*) – input array (3-dimensional)

Returns **out** – transformed array

Return type array_like

freq_meshgrid()
Frequency meshgrid

Returns **out** – temporal and spatial frequency values.

Return type w_vec, ky_vec, kx_vec: quantity arrays

meshgrid()
Spatiotemporal meshgrid

Returns **out** – time, x, and y values.

Return type t_vec, y_vec, x_vec: quantity arrays

positions
Position array

Returns **out** – positions

Return type quantity array

spatial_angular_freqs
Spatial angular frequency array

Returns **out** – spatial angular frequencies

Return type quantity array

spatial_freqs
Spatial frequency array

Returns **out** – spatial frequencies

Return type quantity array

temporal_angular_freqs
Temporal angular frequency array

Returns **out** – temporal angular frequencies

Return type quantity array

temporal_freqs
Temporal frequency array

Returns **out** – temporal frequencies

Return type quantity array

times
Time array

Returns **out** – times

Return type quantity array

CHAPTER 7

Kernels

7.1 Create kernels

New kernels can be defined by creating a closure object where the inner function takes the spatial and/or temporal frequencies as argument, depending on whether it is a spatial, temporal, or spatiotemporal kernel.

An example is shown below:

```
def create_my_kernel():
    def evaluate(w, kx, ky):
        # implementation
    return evaluate
```

7.2 Available kernels

7.2.1 Spatial

- Gaussian
- Difference of Gaussian
- Dirac delta

pylgn.kernels.spatial.**create_delta_ft**(*shift_x=array(0.) * deg*, *shift_y=array(0.) * deg*)

Create delta_ft closure

Parameters

- **shift_x** (*float/quantity scalar*) – Shift in x-direction
- **shift_y** (*float/quantity scalar*) – Shift in y-direction

Returns **out** – Evaluate function

Return type function

pylgn.kernels.spatial.**create_dog_ft**(*A=1*, *a=array(0.62) * deg*, *B=0.85*, *b=array(1.26) * deg*, *dx=array(0.) * deg*, *dy=array(0.) * deg*)

Create Fourier transformed difference of Gaussian function closure

Parameters

- **A** (*float*) – Center peak value
- **a** (*float/quantity scalar*) – Center width
- **B** (*float*) – Surround peak value
- **b** (*float/quantity scalar*) – Surround width
- **dx** (*float/quantity scalar*) – shift in x-direction
- **dy** (*float/quantity scalar*) – shift in y-direction

Returns **out** – Evaluate function

Return type function

pylgn.kernels.spatial.**create_gauss_ft**(*A=1*, *a=array(0.62) * deg*, *dx=array(0.) * deg*, *dy=array(0.) * deg*)

Create Fourier transformed Gaussian function closure.

Parameters

- **A** (*float*) – peak value
- **a** (*float/quantity scalar*) – Width
- **dx** (*float/quantity scalar*) – shift in x-direction
- **dy** (*float/quantity scalar*) – shift in y-direction

Returns **out** – Evaluate function

Return type function

7.2.2 Temporal

- Dirac delta
- Biphasic
- Difference of exponentials

pylgn.kernels.temporal.**create_biphasic_ft**(*phase=array(43.) * ms*, *damping=0.38*, *delay=array(0.) * ms*)

Create Fourier transformed Biphasic closure

Parameters

- **phase** (*float/quantity scalar*) – Delay
- **damping** (*float*) – Damping factor
- **delay** (*float/quantity scalar*) – Delay

Returns **out** – Evaluate function

Return type function

pylgn.kernels.temporal.**create_delta_ft**(*delay=array(0.) * ms*)

Create Fourier transform delta closure

Parameters **delay** (*float/quantity scalar*) – Delay

Returns **out** – Evaluate function

Return type function

pylgn.kernels.temporal.**create_exp_decay_ft**(*tau*, *delay*)

Create Fourier transformed exponential decay closure

Parameters

- **tau** (*float/quantity scalar*) – Time constant
- **delay** (*float/quantity scalar*) – Delay

Returns **out** – Evaluate function

Return type function

CHAPTER 8

Stimulus

8.1 Available stimulus

- Full-field grating
- Patch grating
- Flashing spot
- Natural scenes and movies

pylgn.stimulus.**create_flashing_spot**(*contrast=1*, *patch_diameter=array(1.) * deg*, *delay=array(0.) * ms*, *duration=array(0.) * ms*)

Create flashing spot

Parameters

- **contrast** (*float*) – Contrast value
- **patch_diameter** (*float/quantity scalar*) – Patch size
- **delay** (*float/quantity scalar*) – onset time
- **duration** (*float/quantity scalar*) – duration of flashing spot

Returns **out** – Evaluate function

Return type callable

pylgn.stimulus.**create_flashing_spot_ft**(*contrast=1*, *patch_diameter=array(1.) * deg*, *delay=array(0.) * ms*, *duration=array(0.) * ms*)

Create Fourier transformed flashing spot

Parameters

- **contrast** (*float*) – Contrast value
- **patch_diameter** (*float/quantity scalar*) – Patch size
- **delay** (*float/quantity scalar*) – onset time

- **duration** (*float/quantity scalar*) – duration of flashing spot

Returns **out** – Evaluate function

Return type callable

pylgn.stimulus.**create_fullfield_grating**(*angular_freq=array(0.) * Hz*, *wavenumber=array(0.) * 1/deg*, *orient=array(0.) * deg*, *contrast=1*)

Create full-field grating

Parameters

- **angular_freq** (*float*) – Angular frequency (positive number)
- **wavenumber** (*float/quantity scalar*) – Wavenumber (positive number)
- **orient** (*float/quantity scalar*) – Orientation
- **contrast** (*float*) – Contrast value

Returns **out** – Evaluate function

Return type callable

Notes

Both angular_freq and wavenumber are positive numbers. Use orientation to specify desired direction.

pylgn.stimulus.**create_fullfield_grating_ft**(*angular_freq=array(0.) * Hz*, *wavenumber=array(0.) * 1/deg*, *orient=array(0.) * deg*, *contrast=1*)

Create Fourier transformed full-field grating

Parameters

- **angular_freq** (*float*) – Angular frequency (positive number)
- **wavenumber** (*float/quantity scalar*) – Wavenumber (positive number)
- **orient** (*float/quantity scalar*) – Orientation
- **contrast** (*float*) – Contrast value

Returns **out** – Evaluate function

Return type callable

Notes

Both angular_freq and wavenumber are positive numbers. Use orientation to specify desired direction. The combination of angular_freq, wavenumber, and orient should give w, kx, and ky that exist in function arguments in evaluate function.

pylgn.stimulus.**create_natural_image**(*filenames*, *delay=array(0.) * ms*, *duration=array(0.) * ms*)

Creates natural image stimulus

Parameters

- **filenames** (*list/string*) – path to image(s)
- **delay** (*quantity scalar*) – Onset time
- **duration** (*quantity scalar*)

Returns **out** – Evaluate function

Return type callable

pylgn.stimulus.**create_natural_movie**(*filename*)

Creates natural movie stimulus

Parameters **filename** (*string*) – path to gif

Returns **out** – Evaluate function

Return type callable

pylgn.stimulus.**create_patch_grating**(*angular_freq=array(0.) * Hz, wavenumber=array(0.) * 1/deg, orient=array(0.) * deg, contrast=1, patch_diameter=array(1.) * deg*)

Create patch grating

Parameters

- **angular_freq** (*float*) – Angular frequency (positive number)
- **wavenumber** (*float/quantity scalar*) – Wavenumber (positive number)
- **orient** (*float/quantity scalar*) – Orientation
- **contrast** (*float*) – Contrast value
- **patch_diameter** (*float/quantity scalar*) – Patch size

Returns **out** – Evaluate function

Return type callable

Notes

Both angular_freq and wavenumber are positive numbers. Use orientation to specify desired direction.

pylgn.stimulus.**create_patch_grating_ft**(*angular_freq=array(0.) * Hz, wavenumber=array(0.) * 1/deg, orient=array(0.) * deg, contrast=1, patch_diameter=array(1.) * deg*)

Create Fourier transformed patch grating

Parameters

- **angular_freq** (*float*) – Angular frequency (positive number)
- **wavenumber** (*float/quantity scalar*) – Wavenumber (positive number)
- **orient** (*float/quantity scalar*) – Orientation
- **contrast** (*float*) – Contrast value
- **patch_diameter** (*float/quantity scalar*) – Patch size

Returns **out** – Evaluate function

Return type callable

Notes

Both angular_freq and wavenumber are positive numbers. Use orientation to specify desired direction. The combination of angular_freq, wavenumber, and orient should give w, kx, and ky that exist in function arguments in evaluate function.

CHAPTER 9

Citing pyLGN

If you use pyLGN in your work, please cite:

> Mobarhan MH, Halnes G, Martínez-Cañada P, Hafting T, Fyhn M, Einevoll G. (2018). PLOS Computational Biology 14(5): e1006156. https://doi.org/10.1371/journal.pcbi.1006156.

CHAPTER 10

Where to start

- *Installation*
- *Getting started*

CHAPTER 11

Examples

- *Spatial receptive-field of nonlagged cells in dLGN of cat*.
- *Response of the DoG model to patch grating*.
- *Extended DoG model with cortical feedback*.
- *Using natural scenes and natural movies as stimulus*.
- *Generate spike trains*.

CHAPTER 12

API reference

Python Module Index

p

Index

Symbols

A

C

D

E