

DiPDE: A simulator for population density modeling

<http://alleninstitute.github.io/dipde/>

HBP CodeJam Workshop #7, January 11, 2016
Shrigley Hall Hotel, Manchester, England

Nicholas Cain
Scientist 1



ALLEN INSTITUTE *for*
BRAIN SCIENCE

Overview: DiPDE

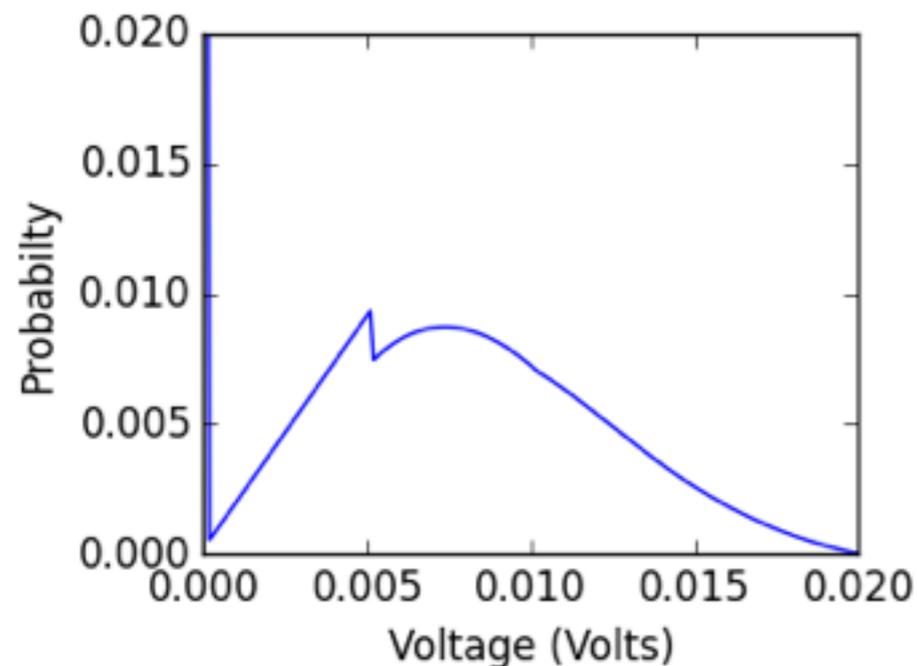
DiPDE (dipde) is a simulation platform for numerically solving the time evolution of coupled networks of neuronal populations. Instead of solving the subthreshold dynamics of individual model leaky-integrate-and-fire (LIF) neurons, dipde models the voltage distribution of a population of neurons with a single population density equation.

In this way, dipde can facilitate the fast exploration of mesoscale (population-level) network topologies, where large populations of neurons are treated as homogeneous with random fine-scale connectivity.

Overview: DiPDE

Goal: Provide a fast, flexible, python-based population statistic simulator for coarse-scale modeling

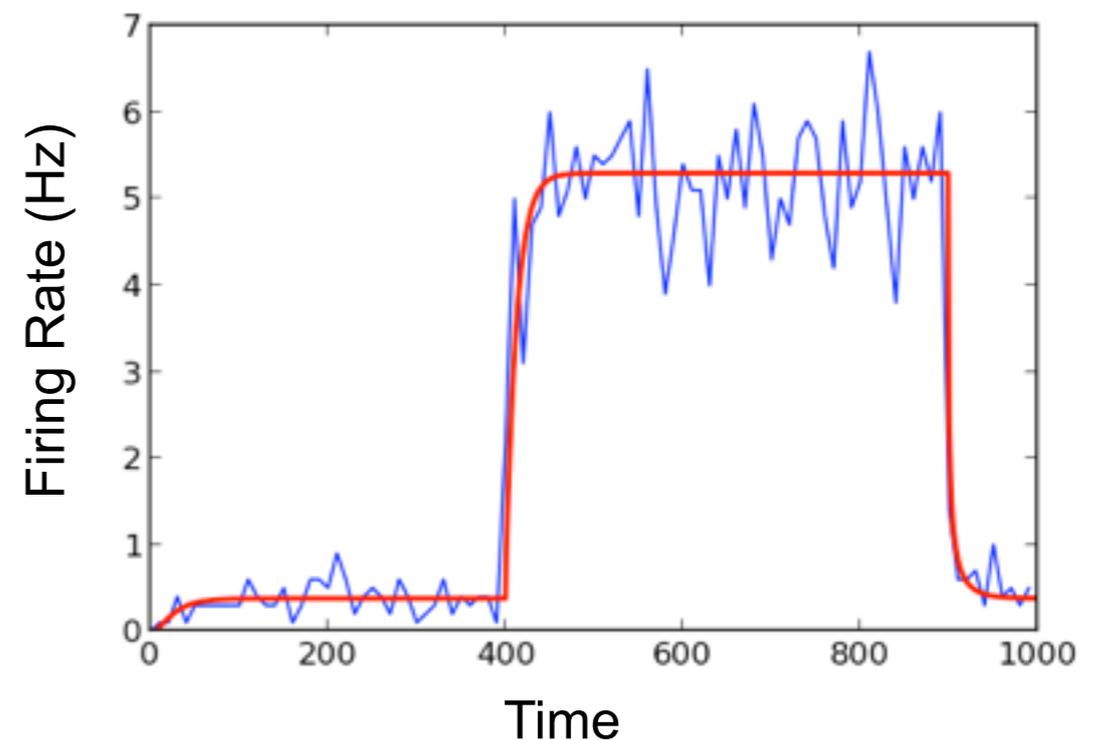
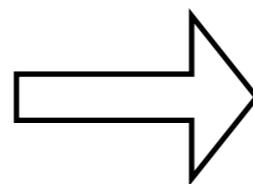
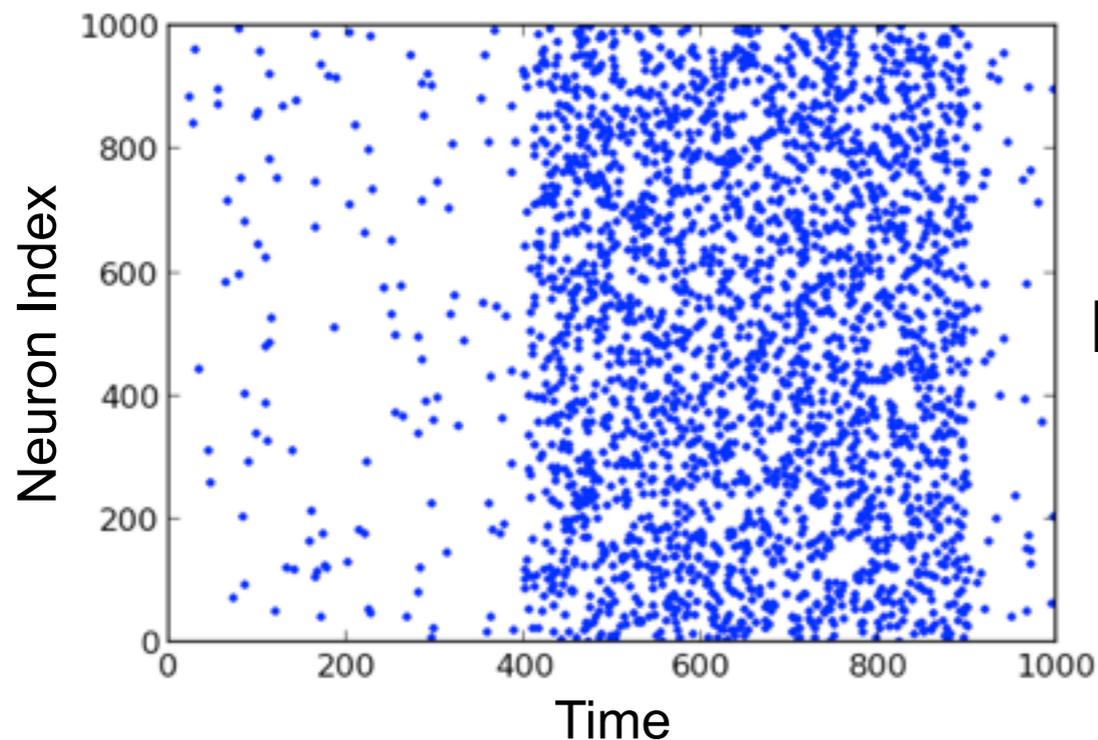
- Solve coupled population density equations (instantaneous synapses)
- Absorbing boundary condition at threshold
- Same mean firing rate as LIF population as N increases
- Exact when representing, ex., firing rate code



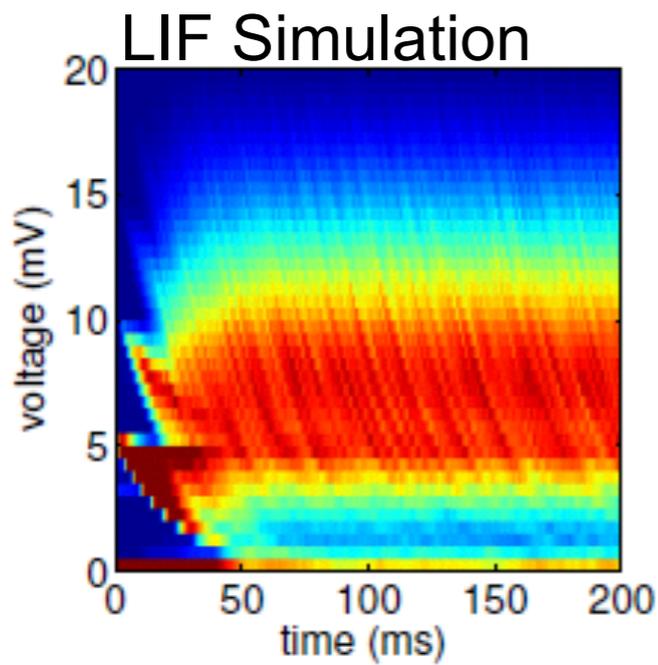
Overview: DiPDE

- Approximate mean firing rate of a LIF population
- Essentially a coupled PDE solver:
 - Boundary condition of source provides drive for target
 - Coupling through synaptic weight/delay distribution
- Allows fast:
 - Stability analysis
 - Stimulus/network topology/parameter exploration
 - Sensitivity analysis

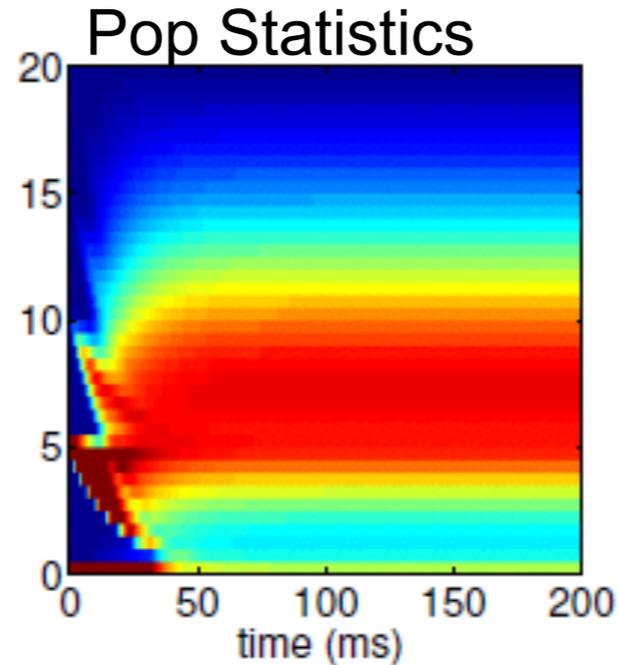
N = 1000 LIF neurons



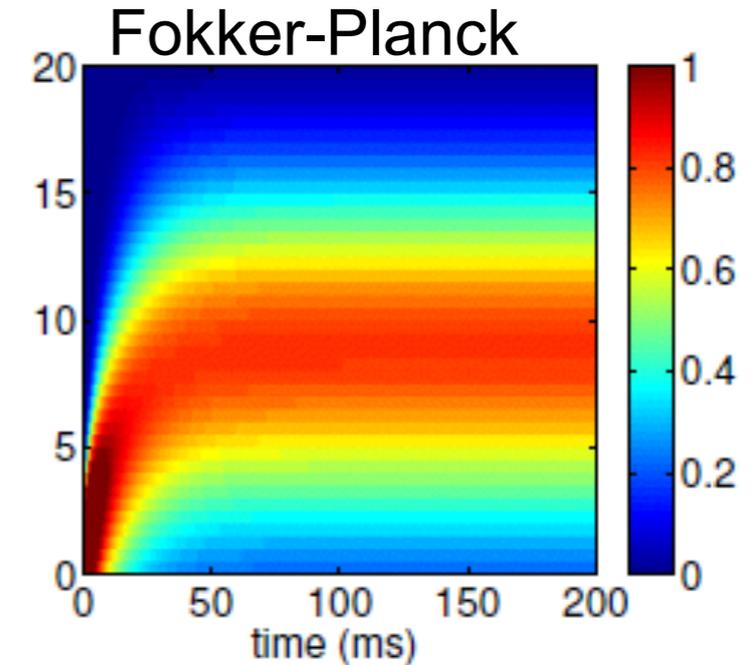
Overview: DiPDE



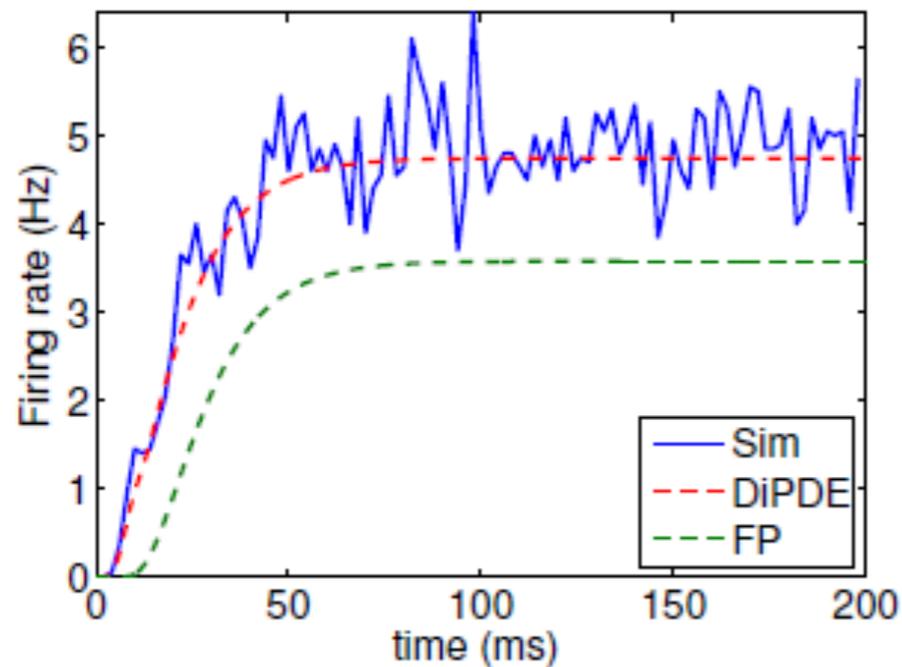
(a)



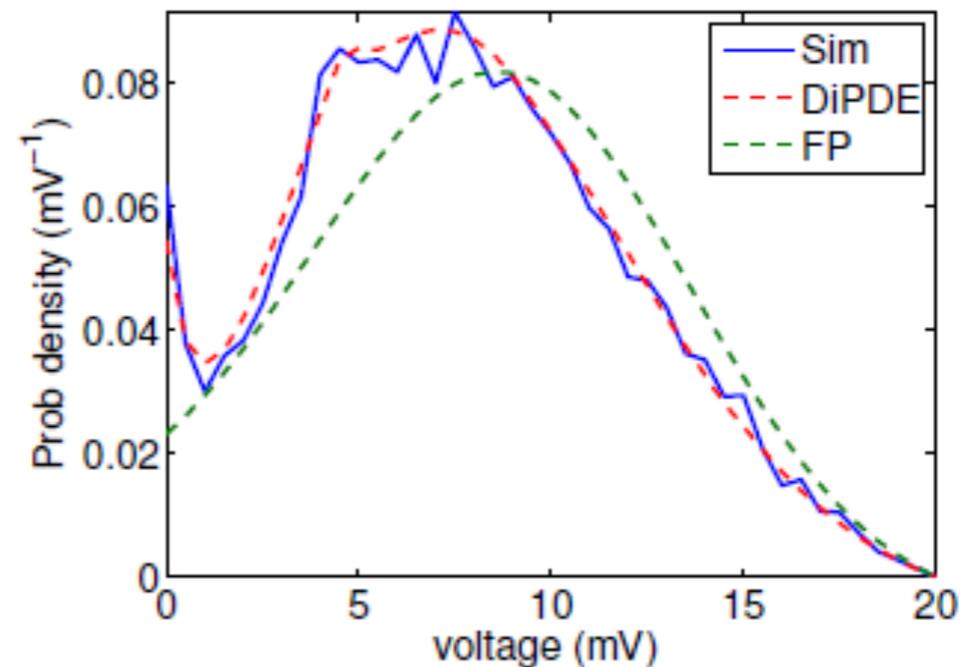
(b)



(c)



(d)



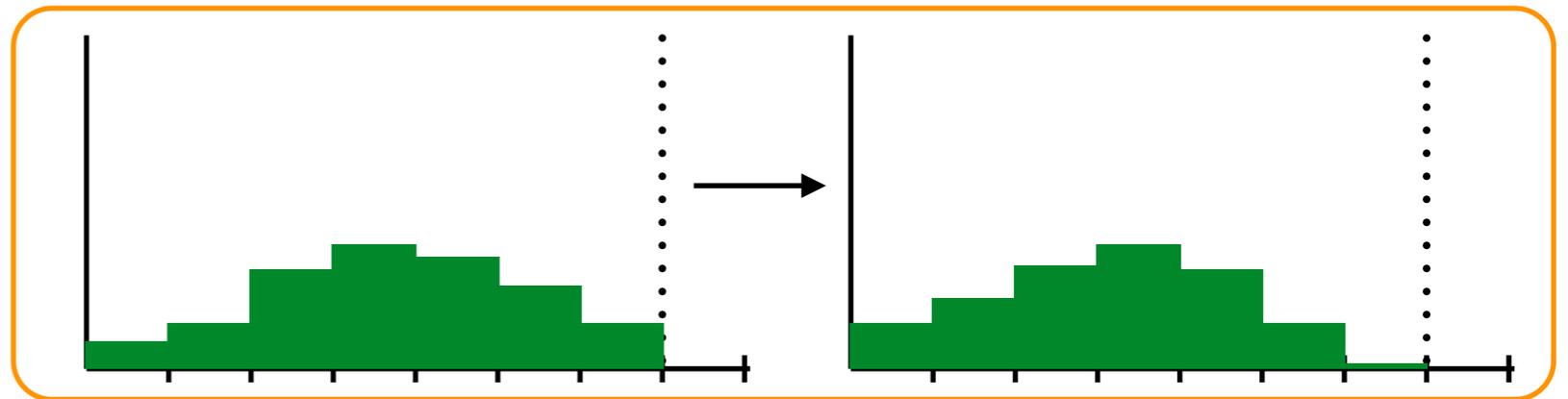
(e)

DiPDE: Summary

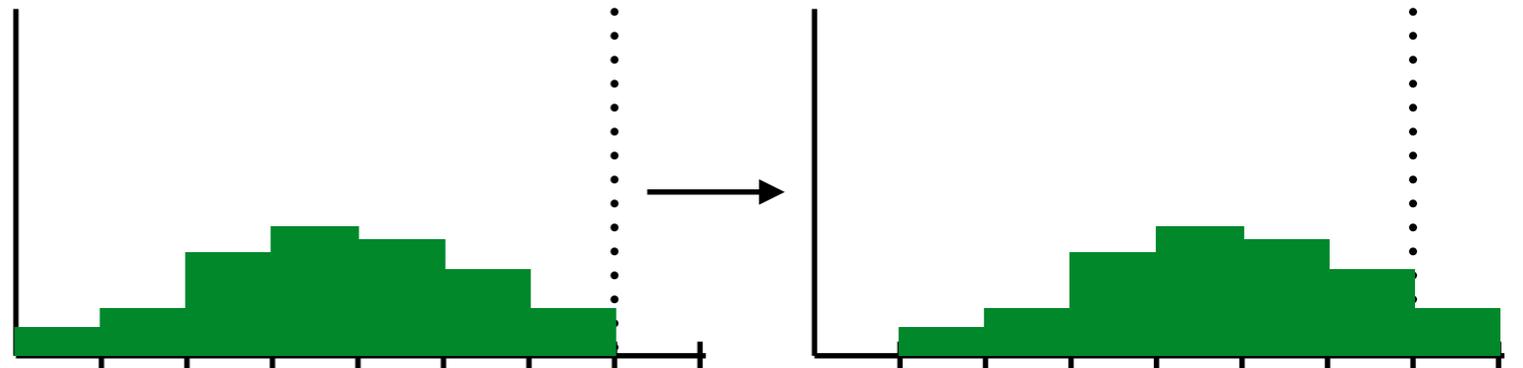
$$\partial_t p(v, t) = \partial_v (L(v)p(v, t)) - f(t)p(v, t) + f(t) \int_{w_1}^{w_2} p(v - w, t)q(w)H(\theta - v + w)dw + j(v, t)$$

$$j(v, t) = f(t) \int_{w_1}^{w_2} H(v)H(w - v)p(v + \theta - w, t)q(w)dw$$

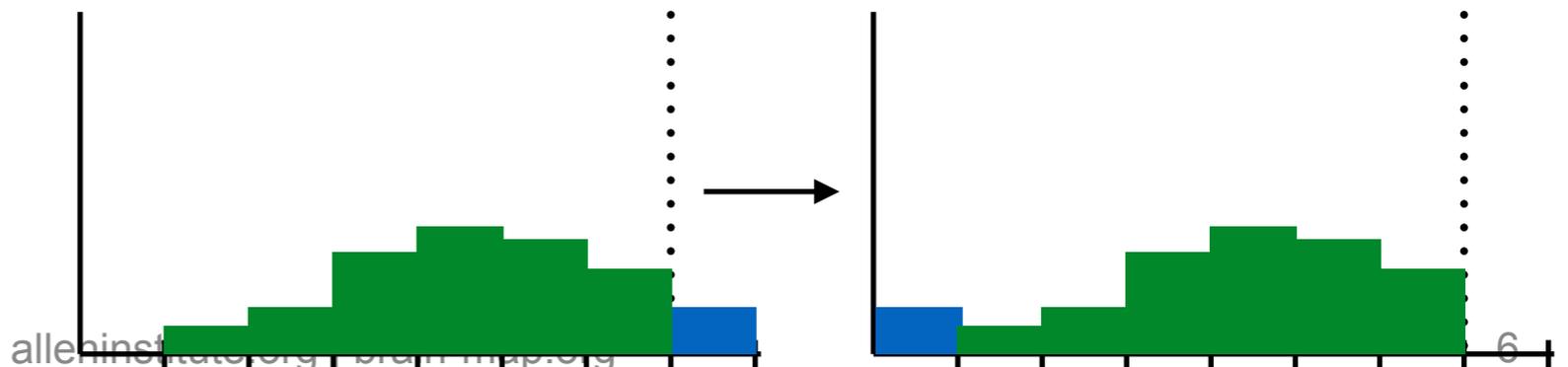
Leak:



Synaptic Activation:



Thresholding:

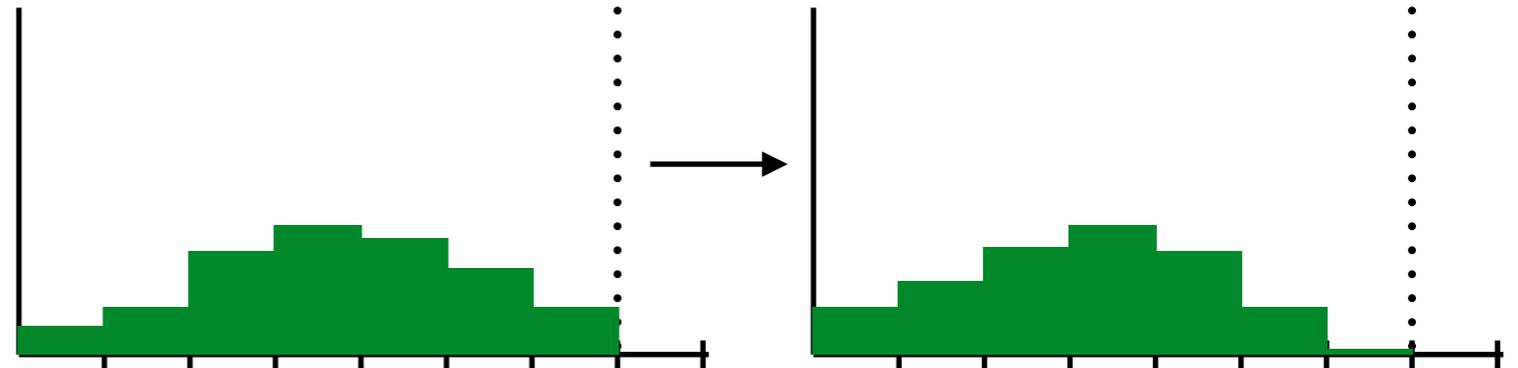


DiPDE: Summary

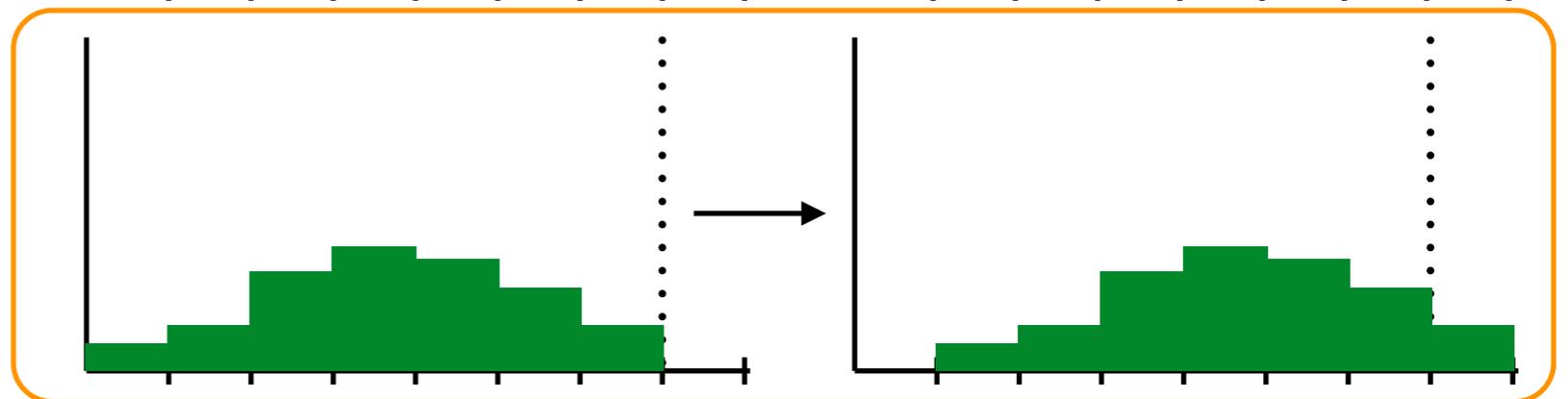
$$\partial_t p(v, t) = \partial_v (L(v)p(v, t)) - f(t)p(v, t) + f(t) \int_{w_1}^{w_2} p(v - w, t)q(w)H(\theta - v + w)dw + j(v, t)$$

$$j(v, t) = f(t) \int_{w_1}^{w_2} H(v)H(w - v)p(v + \theta - w, t)q(w)dw$$

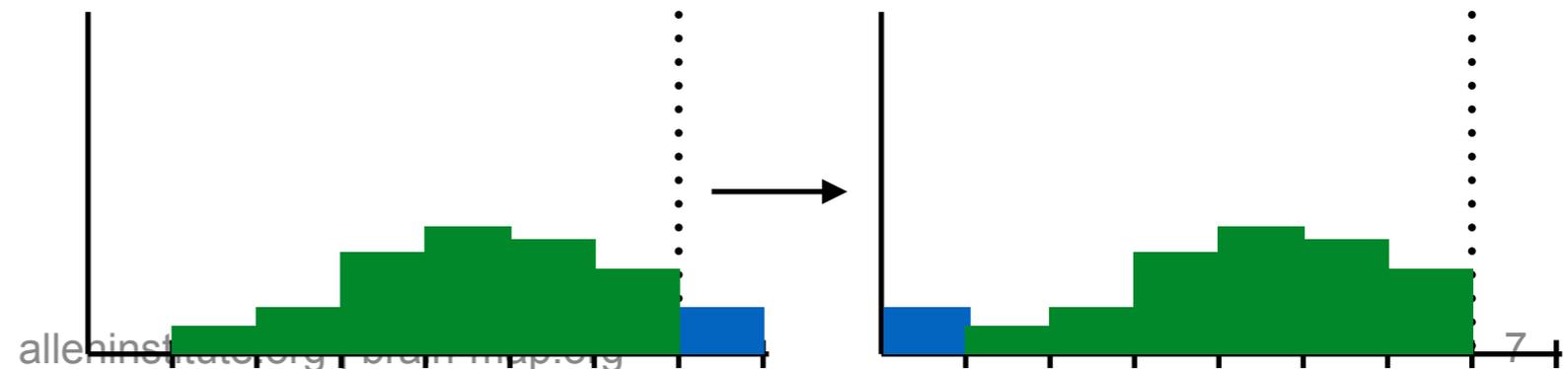
Leak:



Synaptic Activation:



Thresholding:

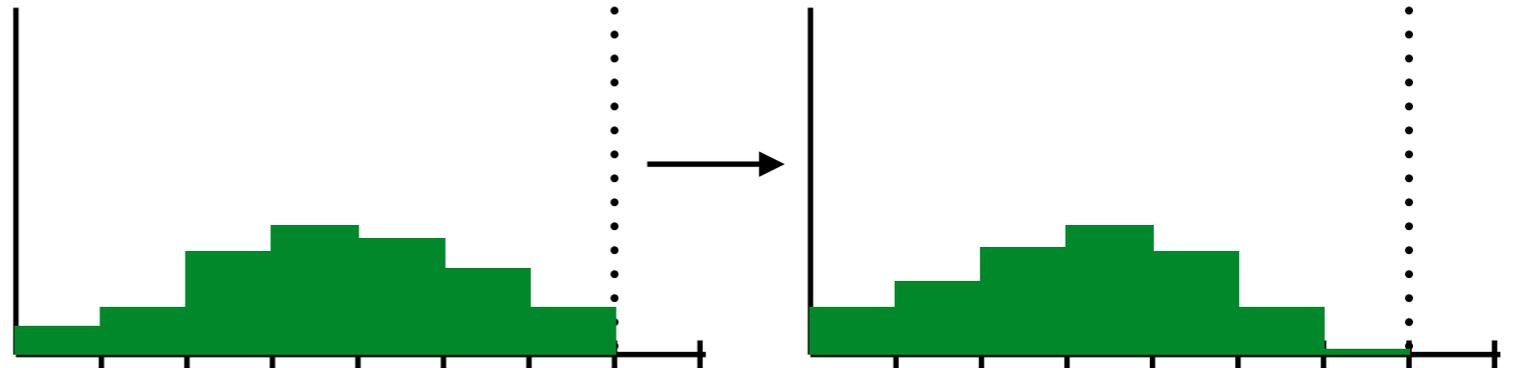


DiPDE: Summary

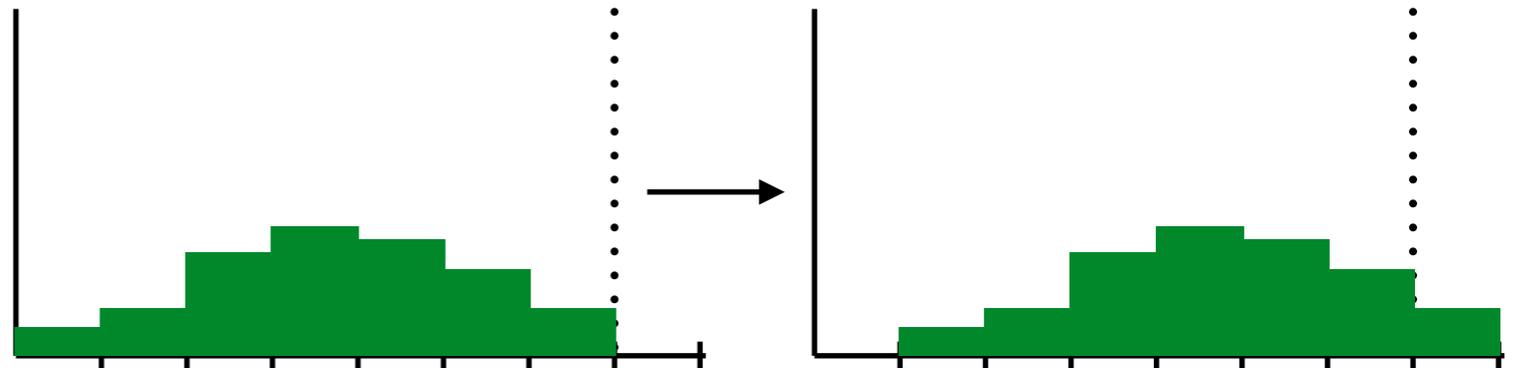
$$\partial_t p(v, t) = \partial_v (L(v)p(v, t)) - f(t)p(v, t) + f(t) \int_{w_1}^{w_2} p(v - w, t)q(w)H(\theta - v + w)dw + j(v, t)$$

$$j(v, t) = f(t) \int_{w_1}^{w_2} H(v)H(w - v)p(v + \theta - w, t)q(w)dw$$

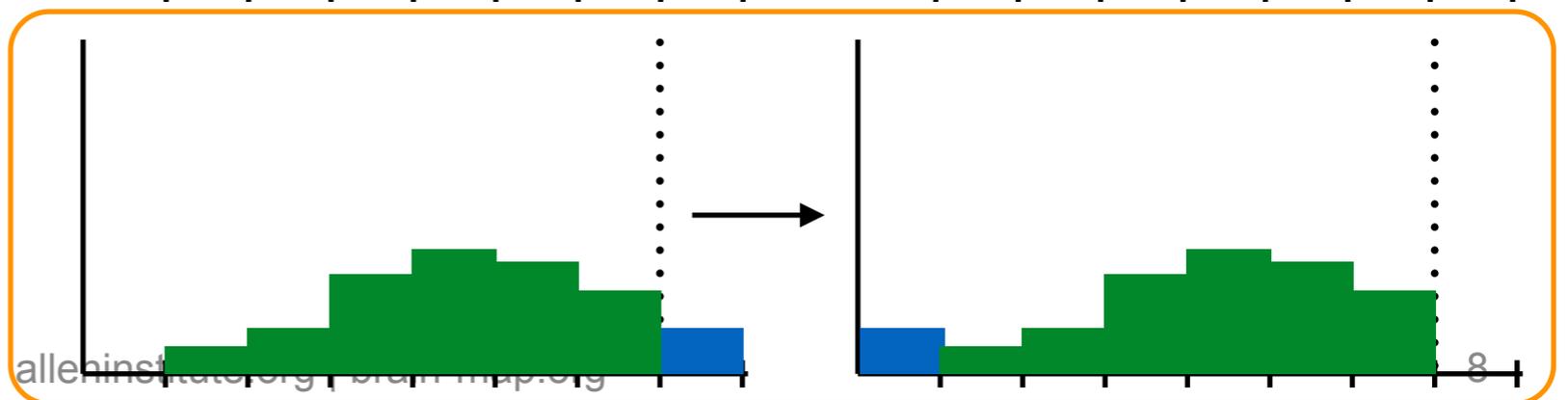
Leak:



Synaptic Activation:

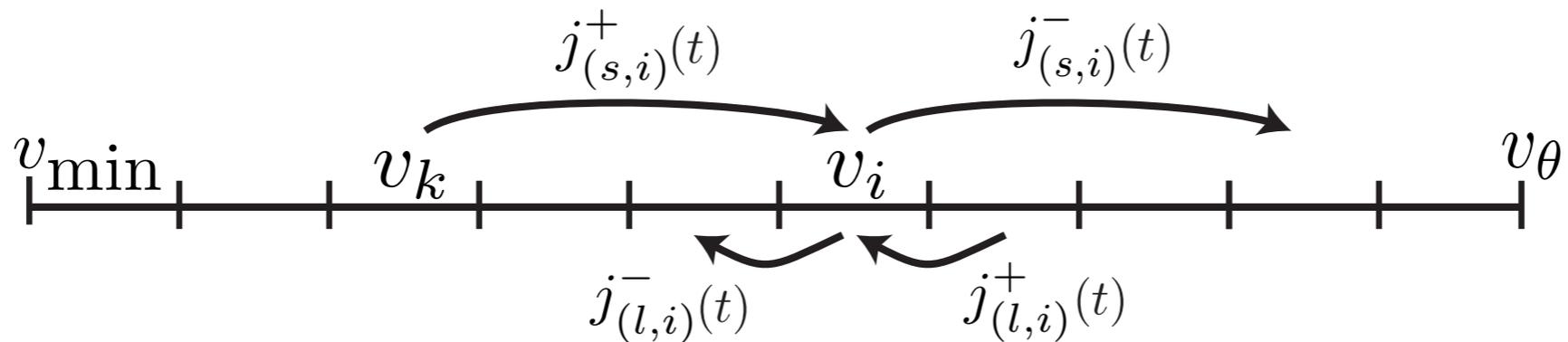


Thresholding:



DiPDE: Summary

- Finite-volume method on the bounded interval
 - Assumes piecewise constant (over dt) recurrent drive
 - Evolve as a continuous time Markov chain



$$\frac{\partial p}{\partial t} = -\frac{\partial J}{\partial v} \quad \rightarrow \quad \frac{\partial p_i}{\partial t} = -\frac{\Delta J_i}{\Delta v_i}$$

$$\begin{aligned} \Delta J_i &= f_{i+\frac{1}{2}} - f_{i-\frac{1}{2}} \\ &= (j_{(s,i)}^- - j_{(l,i)}^+) - (j_{(s,i)}^+ - j_{(l,i)}^-) \end{aligned}$$

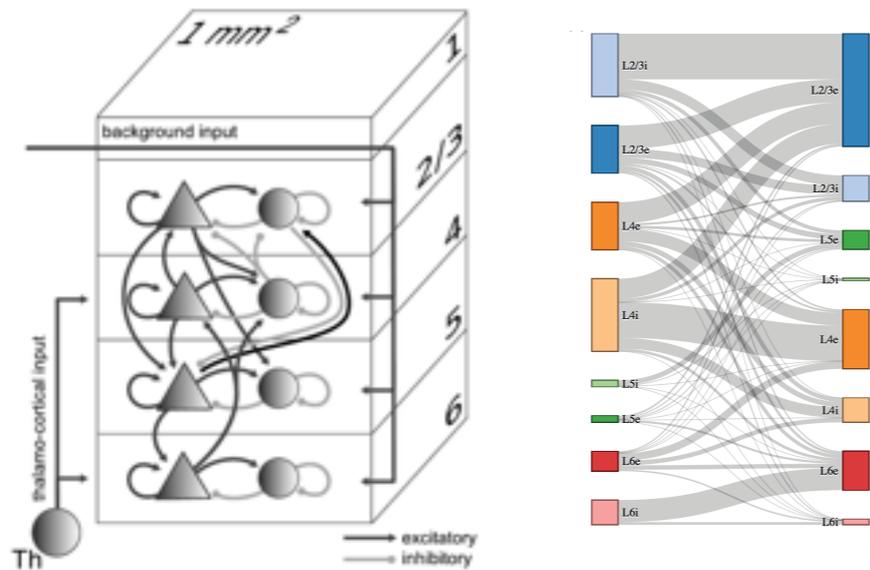
$$j_{(s,i)}^+ = p_k \Delta v_k \lambda_{in}(t)$$

$$j_{(s,i)}^- = p_i \Delta v_i \lambda_{in}(t)$$

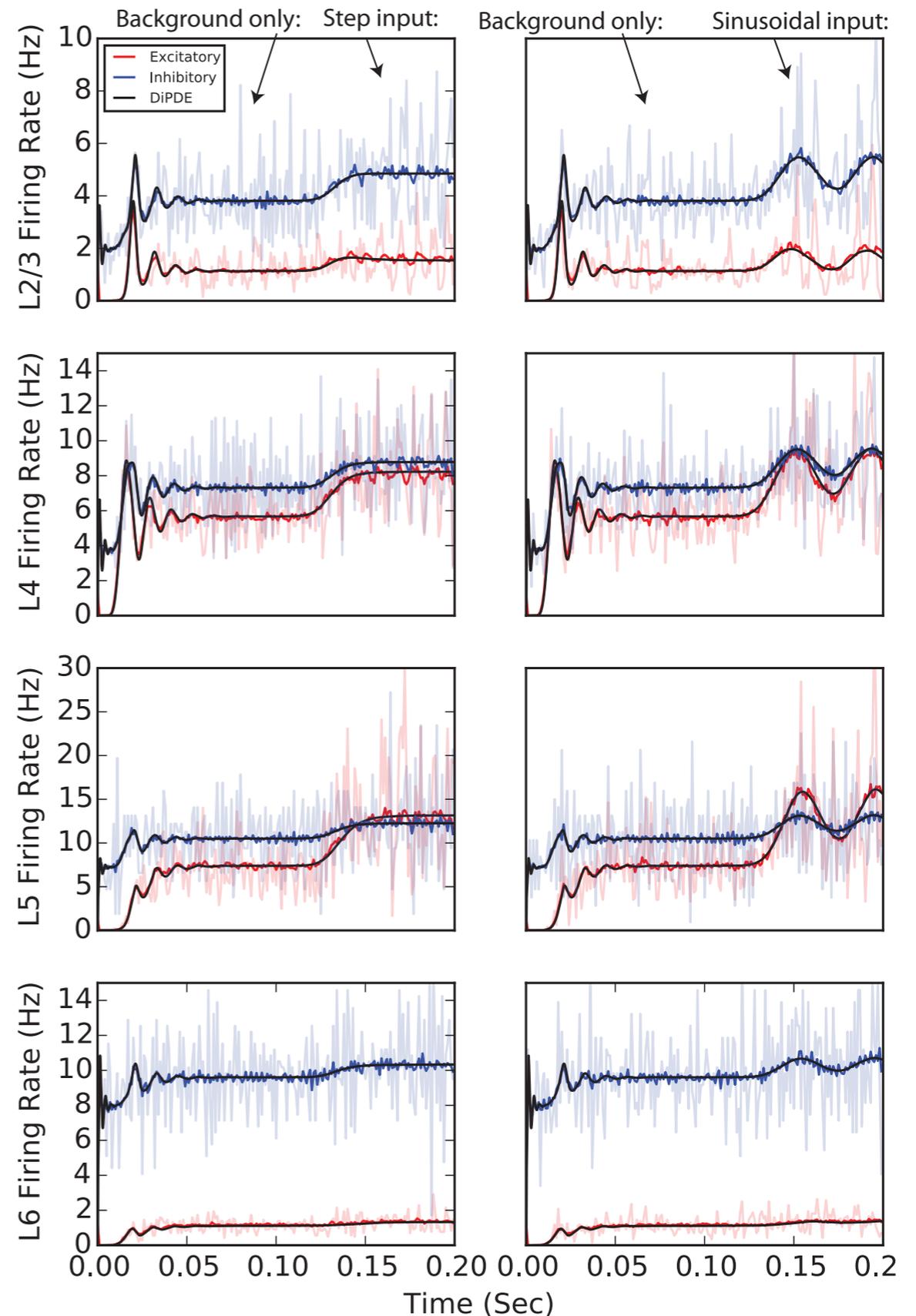
$$j_{(l,i)}^+ = \frac{p_{i+1} v_{i+\frac{1}{2}}}{\tau}$$

$$j_{(l,i)}^- = \frac{p_i v_{i-\frac{1}{2}}}{\tau}$$

- DiPDE well-approximates a simplified cortical column
- Modified version of Potjans and Diesmann (2014)



- Plotted: averaged results from 100 LIF simulations (NEST)
- ~30 second run-time

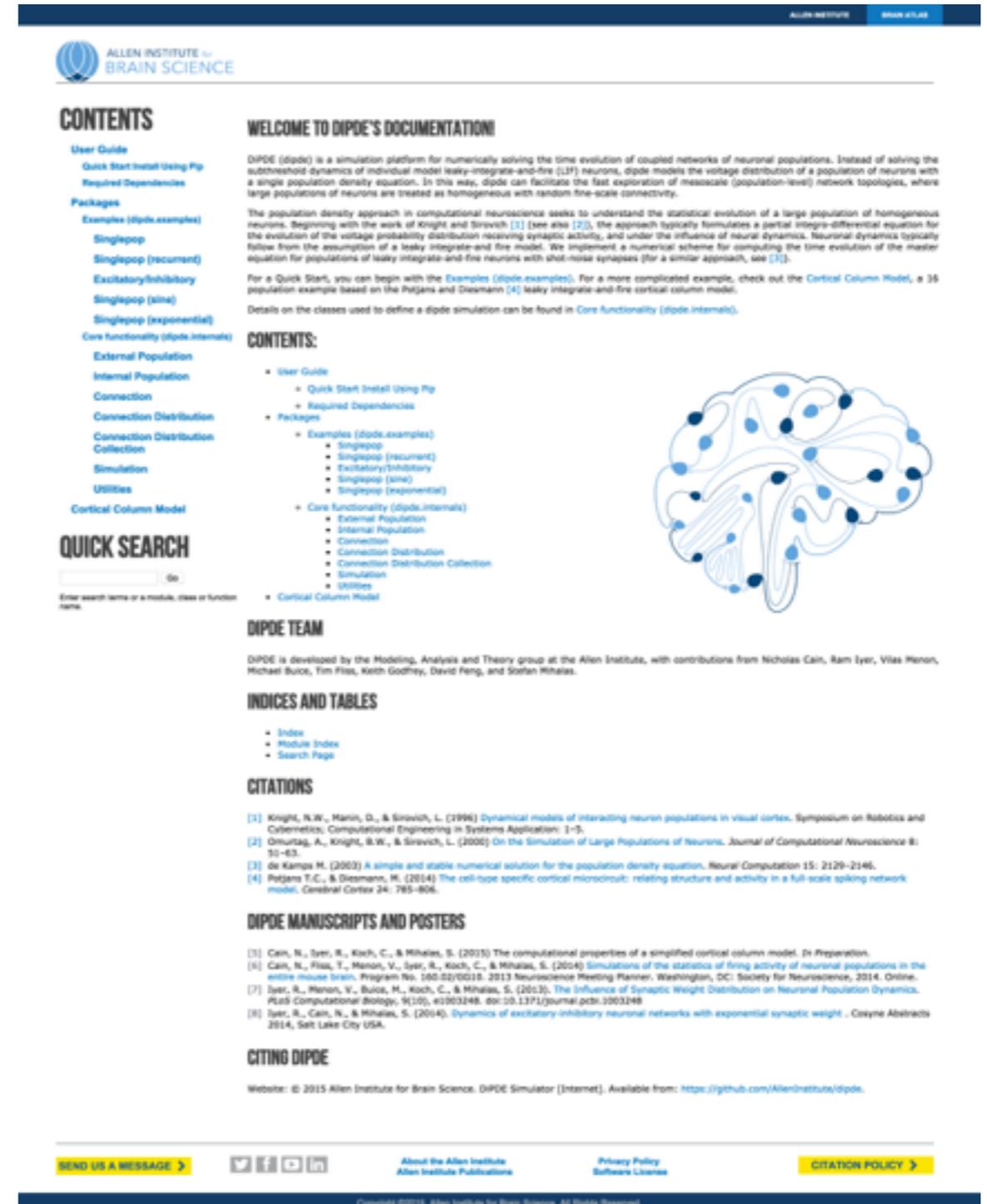


Current Release:

- Current release: 0.1.0
 - <http://alleninstitute.github.io/dipde/>
 - 100% code coverage on tests
 - Pure python; only need numpy/scipy/sympy
- Tutorial features 5 simple examples:
 - <https://goo.gl/kZj2XN>

Documented Features:

- Populations:
 - External: Strings, sympy-functions, python functions
 - Internal: Variable dv/dt /time-step accuracy
- Connections:
 - Synaptic weight distributions
 - Discrete transmission delay distributions
- Simulation: run/pause/continue



ALLEN INSTITUTE FOR BRAIN SCIENCE

CONTENTS

User Guide

- Quick Start (Install Using Pip)
- Required Dependencies

Packages

Examples (dipde-examples)

- Singlepop
- Singlepop (recurrent)
- Excitatory/Inhibitory
- Singlepop (sine)
- Singlepop (exponential)

Core functionality (dipde.internals)

- External Population
- Internal Population
- Connection
- Connection Distribution
- Connection Distribution Collection
- Simulation
- Utilities

Cortical Column Model

QUICK SEARCH

Enter search terms or a module, class or function name.

WELCOME TO DIPDE'S DOCUMENTATION!

DIPDE (dipde) is a simulation platform for numerically solving the time evolution of coupled networks of neuronal populations. Instead of solving the subthreshold dynamics of individual model leaky-integrate-and-fire (LIF) neurons, dipde models the voltage distribution of a population of neurons with a single population density equation. In this way, dipde can facilitate the fast exploration of mesoscale (population-level) network topologies, where large populations of neurons are treated as homogeneous with random fine-scale connectivity.

The population density approach in computational neuroscience seeks to understand the statistical evolution of a large population of homogeneous neurons. Beginning with the work of Knight and Sirovich [1] (see also [2]), the approach typically formulates a partial integro-differential equation for the evolution of the voltage probability distribution receiving synaptic activity, and under the influence of neural dynamics. Neuronal dynamics typically follow from the assumption of a leaky integrate-and-fire model. We implement a numerical scheme for computing the time evolution of the master equation for populations of leaky integrate-and-fire neurons with shot-noise synapses (for a similar approach, see [3]).

For a Quick Start, you can begin with the Examples (dipde-examples). For a more complicated example, check out the Cortical Column Model, a 3D population example based on the Potjans and Diesmann [4] leaky integrate-and-fire cortical column model.

Details on the classes used to define a dipde simulation can be found in Core functionality (dipde.internals).

CONTENTS:

- User Guide
 - Quick Start (Install Using Pip)
 - Required Dependencies
- Packages
 - Examples (dipde-examples)
 - Singlepop
 - Singlepop (recurrent)
 - Excitatory/Inhibitory
 - Singlepop (sine)
 - Singlepop (exponential)
 - Core functionality (dipde.internals)
 - External Population
 - Internal Population
 - Connection
 - Connection Distribution
 - Connection Distribution Collection
 - Simulation
 - Utilities
- Cortical Column Model

DIPDE TEAM

DIPDE is developed by the Modeling, Analysis and Theory group at the Allen Institute, with contributions from Nicholas Cain, Ram Iyer, Vilas Menon, Michael Buice, Tim Fias, Keith Godfrey, David Peng, and Stefan Mihalas.

INDICES AND TABLES

- Index
- Module Index
- Search Page

CITATIONS

[1] Knight, R.W., Marin, D., & Sirovich, L. (1996) Dynamical models of interacting neuron populations in visual cortex. Symposium on Robotics and Cybernetics: Computational Engineering in Systems Application: 3-5.

[2] Omurtag, A., Knight, R.W., & Sirovich, L. (2002) On the Simulation of Large Populations of Neurons. Journal of Computational Neuroscience 8: 51-63.

[3] de Kamps H. (2003) A simple and stable numerical solution for the population density equation. Neural Computation 15: 2129-2146.

[4] Potjans T.C., & Diesmann, M. (2014) The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model. Cerebral Cortex 24: 785-806.

DIPDE MANUSCRIPTS AND POSTERS

[5] Cain, N., Iyer, R., Koch, C., & Mihalas, S. (2013) The computational properties of a simplified cortical column model. In Preparation.

[6] Cain, N., Fias, T., Menon, V., Iyer, R., Koch, C., & Mihalas, S. (2014) Simulations of the statistics of firing activity of neuronal populations in the entire mouse brain. Program No. 150.02/0213. 2013 Neuroscience Meeting Planner. Washington, DC: Society for Neuroscience, 2014. Online.

[7] Iyer, R., Menon, V., Buice, M., Koch, C., & Mihalas, S. (2013). The Influence of Synaptic Weight Distribution on Neuronal Population Dynamics. PLoS Computational Biology, 9(10), e1003248. doi:10.1371/journal.pcbi.1003248

[8] Iyer, R., Cain, N., & Mihalas, S. (2014). Dynamics of excitatory-inhibitory neuronal networks with exponential synaptic weight. Coyote Abstracts 2014, Salt Lake City USA.

CITING DIPDE

Website: © 2015 Allen Institute for Brain Science. DIPDE Simulator [Internet]. Available from: <https://github.com/AllenInstitute/dipde>.

SEND US A MESSAGE > About the Allen Institute Allen Institute Publications Privacy Policy Software Licenses CITATION POLICY >

Copyright ©2015, Allen Institute for Brain Science. All Rights Reserved.

Example: singlepop.ipynb

```
In [12]: import matplotlib.pyplot as plt
from dipde.internals.internalpopulation import InternalPopulation
from dipde.internals.externalpopulation import ExternalPopulation
from dipde.internals.simulation import Simulation
from dipde.internals.connection import Connection as Connection
%matplotlib inline
```

```
In [13]: # Settings:
t0 = 0.
dt = .0001
dv = .0001
tf = .1
tol = 1e-14
verbose = False
```

```
In [14]: # Create and run simulation:
b1 = ExternalPopulation('100', record=True)
i1 = InternalPopulation(v_min=0, v_max=.02, dv=dv, tol=tol)
b1_i1 = Connection(b1, i1, 1, weights=[.005], probs=[1.], delay=0.0)
simulation = Simulation([b1, i1], [b1_i1], verbose=verbose)
simulation.run()
```

```
In [15]: # Visualize:
i1 = simulation.population_list[1]
plt.figure(figsize=(3,3))
plt.plot(i1.t_record, i1.firing_rate_record)
plt.xlim([0,tf])
plt.ylim(ymin=0)
plt.xlabel('Time (s)')
plt.ylabel('Firing Rate (Hz)')
```

```
class Simulation(object):
```

```
def run(self):
```

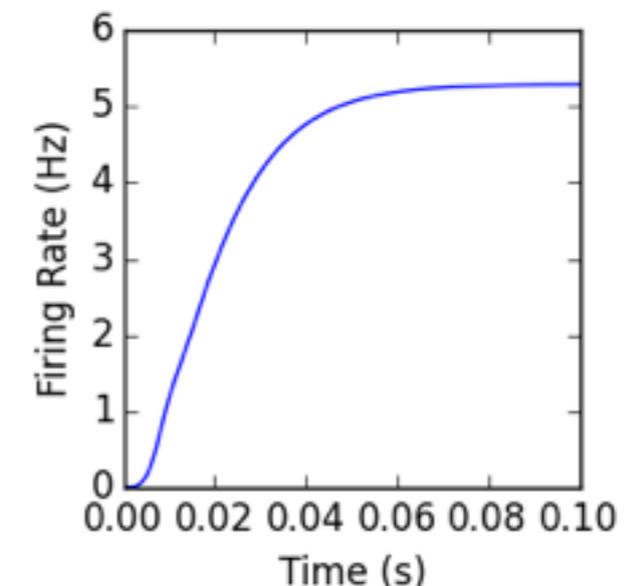
```
for p in self.population_list:
    p.initialize()
```

```
for c in self.connection_list:
    c.initialize()
```

```
while self.t < self.tf:
    self.t += self.dt
```

```
for p in self.population_list:
    p.update()
```

```
for c in self.connection_list:
    c.update()
```

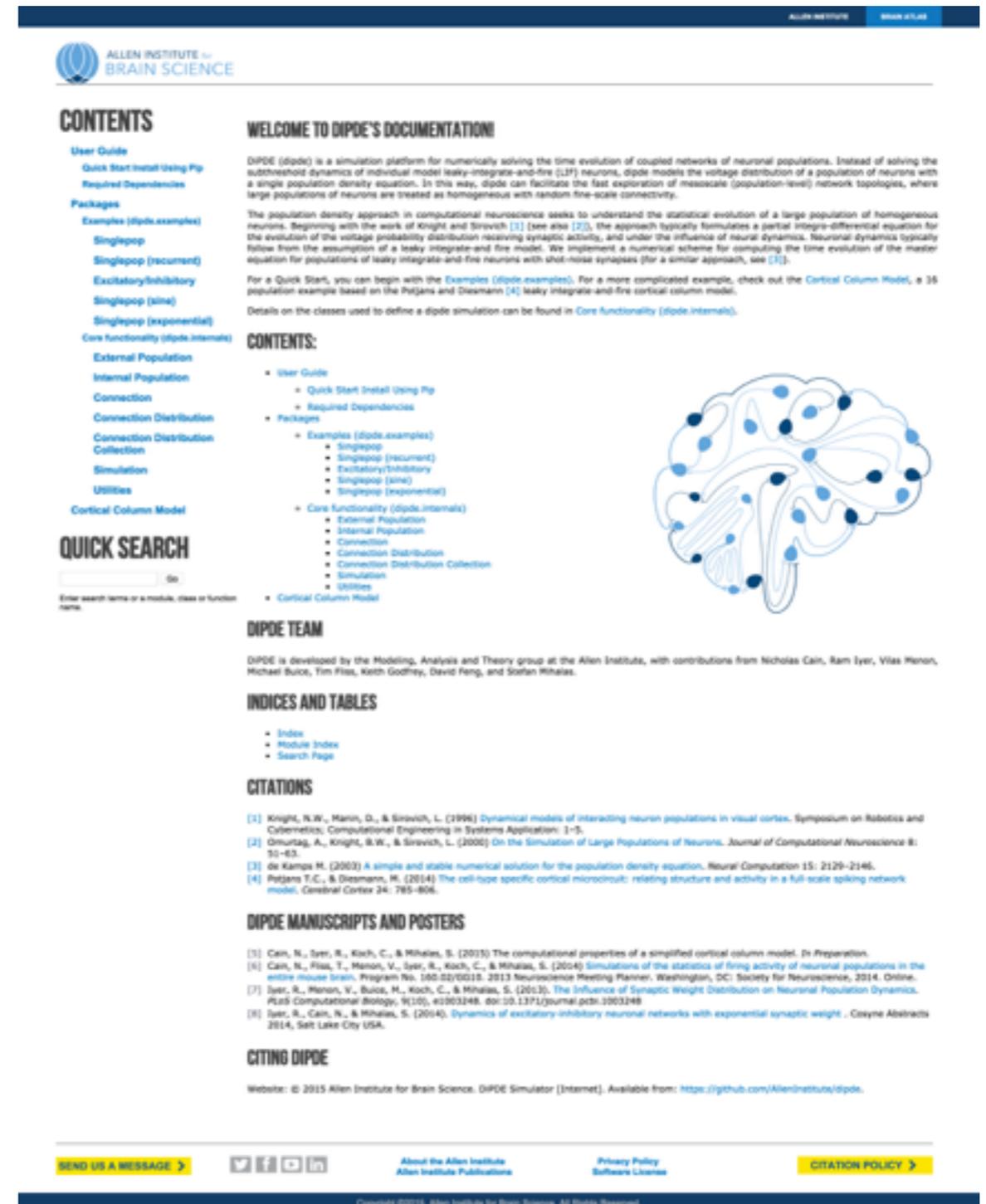


Under Development:

- Next release: 0.2.0 (March 2016)
- https://github.com/nicain/dipde_dev

Features: (completed, debugged, in-progress)

- Distributions of:
 - Synaptic weights
 - Transmission delays
 - Membrane time-constants
- Simple Serialization (JSON)
- Flexible Interface (extend populations)
- ZMQ server/client inputs/outputs
- Run/pause/marshal/unmarshal/continue
- Callbacks on critical functions
- Logging, profiling
- Adapter to NEST/Brian/PyNN
- Algorithmic improvements:
 - Sparse storage
 - (2x-10x) speed up
- Prototype distributed version
- NWB export interface



ALLEN INSTITUTE FOR BRAIN SCIENCE

CONTENTS

User Guide

- Quick Start (Install Using Pip)
- Required Dependencies

Packages

- Examples (dipde-examples)
 - Singlepop
 - Singlepop (recurrent)
 - Excitatory/Inhibitory
 - Singlepop (sine)
 - Singlepop (exponential)
- Core functionality (dipde-internals)
 - External Population
 - Internal Population
 - Connection
 - Connection Distribution
 - Connection Distribution Collection
 - Simulation
 - Utilities
- Cortical Column Model

QUICK SEARCH

Enter search terms or a module, class or function name.

WELCOME TO DIPDE'S DOCUMENTATION!

DIPDE (dipde) is a simulation platform for numerically solving the time evolution of coupled networks of neuronal populations. Instead of solving the subthreshold dynamics of individual model leaky-integrate-and-fire (LIF) neurons, dipde models the voltage distribution of a population of neurons with a single population density equation. In this way, dipde can facilitate the fast exploration of mesoscale (population-level) network topologies, where large populations of neurons are treated as homogeneous with random fine-scale connectivity.

The population density approach in computational neuroscience seeks to understand the statistical evolution of a large population of homogeneous neurons. Beginning with the work of Knight and Sirovich [1] (see also [2]), the approach typically formulates a partial integro-differential equation for the evolution of the voltage probability distribution receiving synaptic activity, and under the influence of neural dynamics. Neuronal dynamics typically follow from the assumption of a leaky integrate-and-fire model. We implement a numerical scheme for computing the time evolution of the master equation for populations of leaky integrate-and-fire neurons with shot-noise synapses (for a similar approach, see [3]).

For a Quick Start, you can begin with the Examples (dipde-examples). For a more complicated example, check out the Cortical Column Model, a 35 population example based on the Pósgans and Diesmann [4] leaky integrate-and-fire cortical column model.

Details on the classes used to define a dipde simulation can be found in Core functionality (dipde-internals).

CONTENTS:

- User Guide
 - Quick Start (Install Using Pip)
 - Required Dependencies
- Packages
 - Examples (dipde-examples)
 - Singlepop
 - Singlepop (recurrent)
 - Excitatory/Inhibitory
 - Singlepop (sine)
 - Singlepop (exponential)
 - Core functionality (dipde-internals)
 - External Population
 - Internal Population
 - Connection
 - Connection Distribution
 - Connection Distribution Collection
 - Simulation
 - Utilities
 - Cortical Column Model

DIPDE TEAM

DIPDE is developed by the Modeling, Analysis and Theory group at the Allen Institute, with contributions from Nicholas Cain, Ram Iyer, Vilas Menon, Michael Buice, Tim Fias, Keith Godfrey, David Peng, and Stefan Mihalas.

INDICES AND TABLES

- Index
- Module Index
- Search Page

CITATIONS

[1] Knight, R.W., Marin, D., & Sirovich, L. (1996) Dynamical models of interacting neuron populations in visual cortex. Symposium on Robotics and Cybernetics, Computational Engineering in Systems Application: 3-5.

[2] Omurtag, A., Knight, R.W., & Sirovich, L. (2002) On the Simulation of Large Populations of Neurons. Journal of Computational Neuroscience 8: 51-63.

[3] de Kamps H. (2003) A simple and stable numerical solution for the population density equation. Neural Computation 15: 2129-2146.

[4] Pósgans T.C., & Diesmann, M. (2014) The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model. Cerebral Cortex 24: 785-806.

DIPDE MANUSCRIPTS AND POSTERS

[5] Cain, N., Iyer, R., Koch, C., & Mihalas, S. (2015) The computational properties of a simplified cortical column model. In Preparation.

[6] Cain, N., Fias, T., Menon, V., Iyer, R., Koch, C., & Mihalas, S. (2014) Simulations of the statistics of firing activity of neuronal populations in the entire mouse brain. Program No. 150.02/0213. 2013 Neuroscience Meeting Planner. Washington, DC: Society for Neuroscience, 2014. Online.

[7] Iyer, R., Menon, V., Buice, M., Koch, C., & Mihalas, S. (2013). The Influence of Synaptic Weight Distribution on Neuronal Population Dynamics. PLoS Computational Biology, 9(10), e1003248. doi:10.1371/journal.pcbi.1003248

[8] Iyer, R., Cain, N., & Mihalas, S. (2014). Dynamics of excitatory-inhibitory neuronal networks with exponential synaptic weight. Coyote Abstracts 2014, Salt Lake City USA.

CITING DIPDE

Website: © 2015 Allen Institute for Brain Science. DIPDE Simulator [Internet]. Available from: <https://github.com/AllenInstitute/dipde>.

SEND US A MESSAGE > About the Allen Institute Allen Institute Publications Privacy Policy Software Licenses CITATION POLICY >

Copyright ©2015, Allen Institute for Brain Science. All Rights Reserved.

Code Example: Distributions

- In 0.2.0, the following are all equivalent ways of specifying a connection distribution:

```
c = Connection(source, target, 1, weights=.005)
c = Connection(source, target, 1, weights=(.005, 1.))
c = Connection(source, target, 1, weights=sps.rv_discrete(values=(.005, 1.)))

c = Connection(source, target, 1, weights=sps.expon(0, .005))
c = Connection(source, target, 1, weights=(sps.expon(0, .005), 201))
c = Connection(source, target, 1, weights={'distribution': 'exponential', 'lambda': .005})
```

Code Example: Interface

- Basic interface to create (firing rate) populations

```
class PopulationInterface(object):
    '''Abstract Base Class for source populations'''

    def initialize(self):
        '''Override with behavior that sets an initial value'''
        self.set_curr_firing_rate(None)

    def update(self):
        '''Override with behavior that gets called once per time step'''
        logger.debug('GID(%s) Firing rate: %s' % (self.gid, self.curr_firing_rate))

    def set_curr_firing_rate(self, curr_firing_rate):
        '''Call to make "curr_firing_rate" visible to other populations.
        Typically invoked once at initialization, and once in update'''
        self._curr_firing_rate = curr_firing_rate

    @property
    def t(self): return self.simulation.t

    @property
    def dt(self): return self.simulation.dt

    @property
    def gid(self): return self.simulation.gid_dict[self]

    @property
    def curr_firing_rate(self): return self._curr_firing_rate

    @property
    def source_connection_list(self): return [c for c in self.simulation.connection_list if c.target == self]

    @property
    def source_firing_rate_dict(self):
        return dict((c.source.gid, self.simulation.get_curr_firing_rate(c.source.gid)) for c in self.source_connection_list)
```

Code Example: ZMQ REQ/REP Servers

- Callable that can be used as the firing_rate arg of an ExternalPopulation

```
class RequestFiringRate(object):

    def __init__(self, port):

        self.port = port
        self.socket = context.socket(zmq.REQ)
        self.socket.connect("tcp://localhost:%s" % port)

    def __call__(self, t):
        self.socket.send('%s' % t)
        message = self.socket.recv_multipart()
        return float(message[0])

class ReplyFiringRateServer(object):

    def __init__(self, port, reply_function):
        self.port = port
        self.reply_function = reply_function
        self.socket = context.socket(zmq.REP)
        self.socket.bind("tcp://*:%s" % self.port)

    def run(self):

        while True:
            message = self.socket.recv()
            if message == 'SHUTDOWN':
                break
            requested_t = float(message)
            self.socket.send_multipart([b"%s" % self.reply_function(requested_t)])
            self.socket.send('DOWN')
```

Code Example: NEST Adapter

- Construct an analogous NEST simulation from a dipde simulation

```
def get_kernel(dt=.0001, tf=.1, seed=None, number_of_processors=2, verbose=True):
    if seed is None: seed = np.random.randint(1,100000)
    import nest as kernel
    kernel.ResetKernel()
    kernel.SetKernelStatus({"local_num_threads": number_of_processors})
    N_vp = kernel.GetKernelStatus(['total_num_virtual_procs'])[0]
    kernel.SetKernelStatus({'grng_seed' : seed+N_vp})
    kernel.SetKernelStatus({'rng_seeds' : range(seed+N_vp+1, seed+2*N_vp+1)})
    kernel.SetKernelStatus({"resolution": dt*1000, "print_time": verbose})
    return kernel

class PoissonPopulation(object):
    def __init__(self, name, firing_rate, number_of_neurons, kernel, start=0.):
        self.name = name
        self.firing_rate = firing_rate
        self.number_of_neurons = number_of_neurons
        self.gids = kernel.Create("poisson_generator", number_of_neurons, params={"rate": float(firing_rate),
                                                                                  'start':float(start)/.001})

class IAFPSCDeltaPopulation(object):
    def __init__(self, name, number_of_neurons, kernel, tau_refrac=0.):
        self.name = name
        if tau_refrac == 0.: tau_refrac = kernel.GetKernelStatus("resolution")/1000
        curr_neuron_params= { "V_reset" : 0., "tau_m" : 10.,
                              "C_m" : 250., "V_th" : 15.,
                              "t_ref" : tau_refrac*1000, "V_m" : 0.,
                              "E_L" : 0.}
        self.gids = kernel.Create("iaf_psc_delta", number_of_neurons, params=curr_neuron_params)
```

Goals for This CodeJam:

1. Meet as many new people and technologies as I can
2. Have fun writing code with all of you
3. Get feedback on the technical approaches I am taking
4. Help anyone who is interested to learn more about dipde
5. Work in interfacing any/all AIBS code and data formats with community standards. (I do more than just dipde)
6. Work on model construction tools, and description formats/adapters
7. Interface dipde with any relevant visualization tools
8. Get help prioritizing features for the future dipde

THANKS!



THANK YOU

We wish to thank the Allen Institute for Brain Science founders, Paul G. Allen and Jody Allen, for their vision, encouragement and support.

ALLENINSTITUTE.ORG
BRAIN-MAP.ORG



ALLEN INSTITUTE *for*
BRAIN SCIENCE