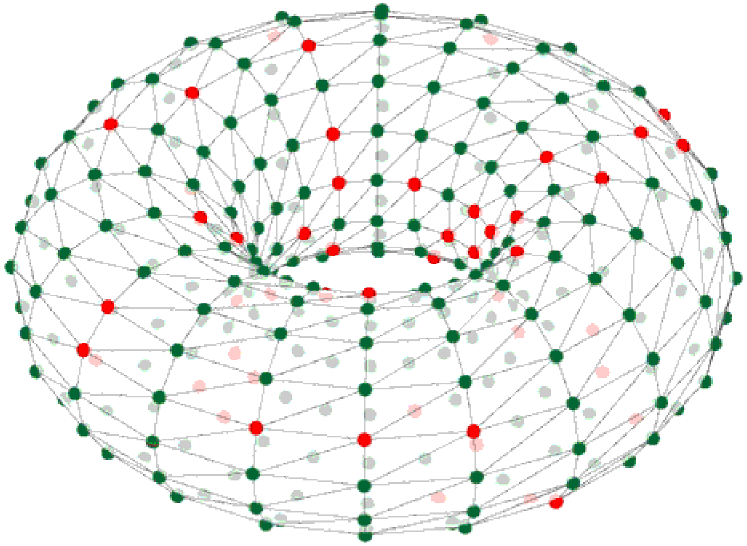
Adding New Neuron Models



Michael Hopkins and Andrew Rowley

SpiNNaker Workshop
September 2015

Required code separation

- Any new neuron model requires both C and Python code
- C code makes the actual executable (on SpiNNaker), Python code configures the setup and load phases (on the host)
- These are separate but <u>must be</u> perfectly coordinated
- In almost all cases, the C code will be solving an ODE which describes how the neuron state evolves over time and in response to input

We will first describe the C requirements...

Data Structures and Parameters

- The parameters and state of your neuron at any point in time need to be stored in memory
- For each neuron, the C header defines the ordering and size of each stored value
- The C types can be standard integer and floating-point, or ISO draft standard fixed-point, as required (see later talk *Maths & fixed-point libraries*)
- There is also one global data structure which services all neurons on a core

So here is an example using the Izhikevich neuron...

Specific neuron model – data structure

```c
#include <neuron-model.h>

//  Izhikevic neuron data structure

typedef struct neuron_t {

// 'fixed' parameters - abstract units
    REAL    A;
    REAL    B;
    REAL    C;
    REAL    D;

// variable-state parameters
    REAL    V;          // nominally in [mV]
    REAL    U;

// offset current [nA]
    REAL    I_offset;

// private variable used internally in C code
    REAL    this_h;

} neuron_t;

...
```

Global data structure

```
...

/*
    Global data structure defined in neuron_model_izh_curr_impl.h
*/

typedef struct global_neuron_params_t {

// Machine time step in milliseconds
   REAL   machine_timestep_ms;

} global_neuron_params_t;
```

Implementing the state update

- Neuron models are typically described as systems of initial value ODEs
- At each time step, the internal state of each neuron needs to be updated in response to inherent dynamics and synaptic input
- There are many ways to achieve this; there will usually be a 'best approach' (in terms of balance between accuracy & efficiency) for each neuron model
- An upcoming paper gives a lot more detail: Hopkins & Furber (2015), “Accuracy and Efficiency in Fixed-Point Neural ODE Solvers” , *Neural Computation* **27**, 1–35.
- The key function will always be *neuron_model_state_update*(); the other functions are mainly to support this and allow debugging etc.

Continuing the example by describing the key interfaces...

Neuron model API

```
// pointer to a neuron data type - used in all access operations
typedef struct neuron_t* neuron_pointer_t;


// set the global neuron parameters
void neuron_model_set_global_neuron_params(
    global_neuron_params_pointer_t params);


// converts raw value from ring buffer into correctly scaled input
static input_t neuron_model_convert_input(input_t input);


// key function in timer loop that updates neuron state vars and returns spike
// state
bool neuron_model_state_update(input_t exc_input, input_t inh_input,
                               input_t external_bias, neuron_pointer_t neuron);


// return membrane voltage (= first state variable) for a given neuron
state_t neuron_model_get_membrane_voltage(restrict neuron_pointer_t neuron);


// print out neuron definition and state variables
void neuron_model_print(restrict neuron_pointer_t neuron);
```

Specific neuron model – key functions

```
/* simplified version of Izhikevic neuron code */

// make the discrete changes to state after a spike has occurred
static inline void _neuron_discrete_changes(neuron_pointer_t neuron) {

    neuron->V  = neuron->C;     // reset membrane voltage
    neuron->U += neuron->D;     // offset 2nd state variable
}

// key function in timer loop that updates neuron state vars and returns spike state
bool neuron_model_state_update(input_t exc_input, input_t inh_input,
                               input_t external_bias, neuron_pointer_t neuron) {
// collect inputs
    input_this_timestep = exc_input - inh_input + external_bias + neuron->I_offset;

// most balanced ESR update found so far
    _rk2_kernel_midpoint( neuron->this_h, neuron, input_this_timestep );

// test for threshold crossing
    bool spike = REAL_COMPARE( neuron->V, >=, V_threshold );

    If ( spike ) {
        _neuron_discrete_changes( neuron );

// simple threshold correction - next timestep (only) gets a bump
        neuron->this_h = global_params->machine_timestep_ms * SIMPLE_TQ_OFFSET;
     } else {
        neuron->this_h = global_params->machine_timestep_ms;
     }
    return spike;
}
```

Makefile

```
APP = my_model_curr_exp
```

Makefile

```
APP = my_model_curr_exp

# This is the folder where things will be built (this will be created)
BUILD_DIR = build/
```

Makefile

```
APP = my_model_curr_exp

# This is the folder where things will be built (this will be created)
BUILD_DIR = build/

# This is the list of objects that will make up your neural model, as well as
# the synaptic plasticity type (or lack of if using static_impl)
# TODO: Add any extra objects to be compiled
MODEL_OBJS = $(EXTRA_SRC_DIR)/neuron/models/neuron_model_my_model_curr_exp.o \
             $(SOURCE_DIRS)/neuron/plasticity/synapse_dynamics_static_impl.o
```

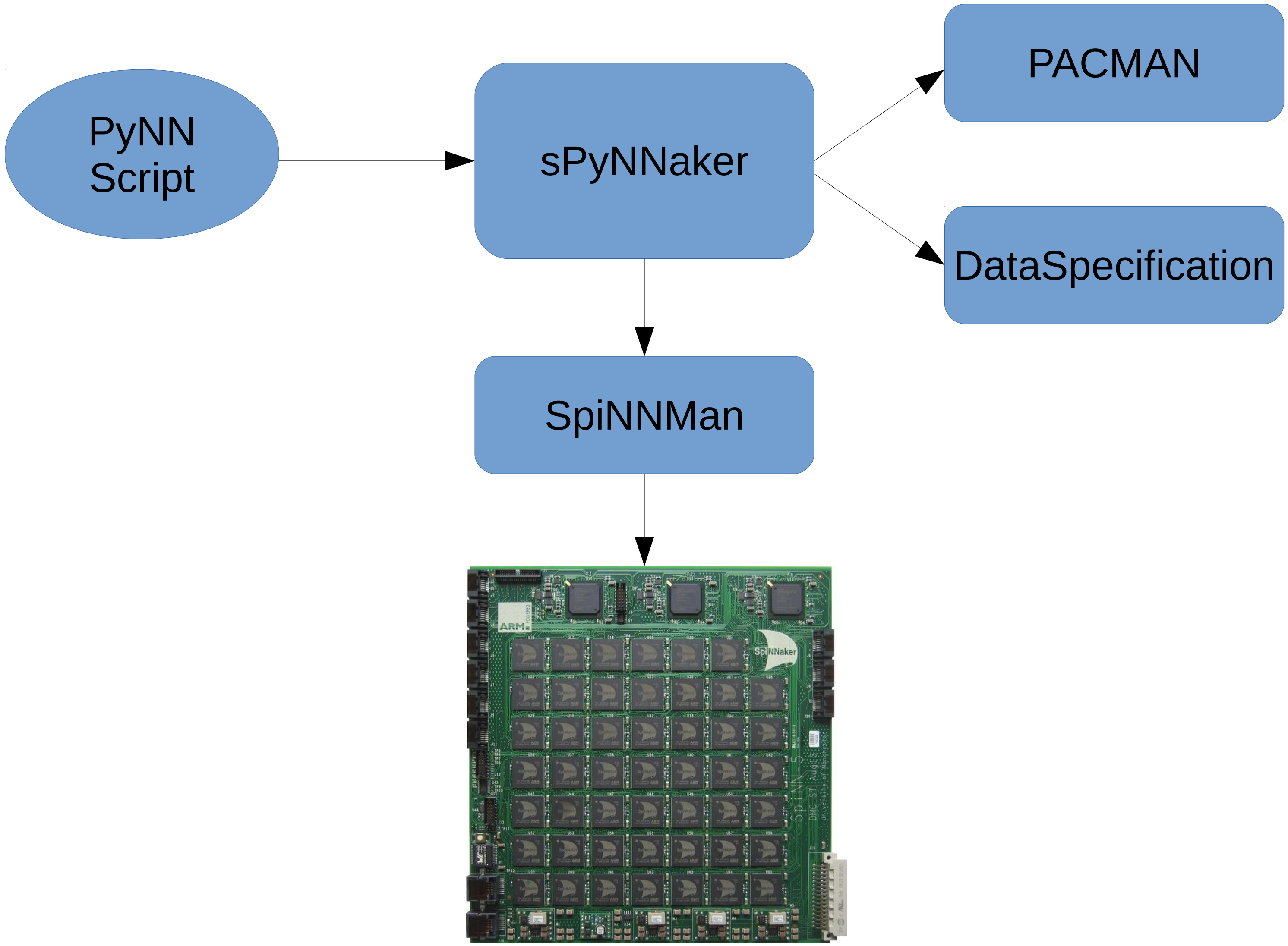
Makefile

```
APP = my_model_curr_exp

# This is the folder where things will be built (this will be created)
BUILD_DIR = build/

# This is the list of objects that will make up your neural model, as well as
# the synaptic plasticity type (or lack of if using static_impl)
# TODO: Add any extra objects to be compiled
MODEL_OBJS = $(EXTRA_SRC_DIR)/neuron/models/neuron_model_my_model_curr_exp.o \
             $(SOURCE_DIRS)/neuron/plasticity/synapse_dynamics_static_impl.o

# This is the header of the neuron model, containing the definintion of
# neuron_t
# TODO: Ensure this matches your neuron model header name
NEURON_MODEL_H = $(EXTRA_SRC_DIR)/neuron/models/neuron_model_my_model_curr_exp.h
```

Makefile

```
APP = my_model_curr_exp

# This is the folder where things will be built (this will be created)
BUILD_DIR = build/

# This is the list of objects that will make up your neural model, as well as
# the synaptic plasticity type (or lack of if using static_impl)
# TODO: Add any extra objects to be compiled
MODEL_OBJS = $(EXTRA_SRC_DIR)/neuron/models/neuron_model_my_model_curr_exp.o \
             $(SOURCE_DIRS)/neuron/plasticity/synapse_dynamics_static_impl.o

# This is the header of the neuron model, containing the definintion of
# neuron_t
# TODO: Ensure this matches your neuron model header name
NEURON_MODEL_H = $(EXTRA_SRC_DIR)/neuron/models/neuron_model_my_model_curr_exp.h

# This is the header containing the synapse shaping type
# (exponential in this case)
# TODO: Ensure that this is the desired shaping
SYNAPSE_TYPE_H = $(SOURCE_DIRS)/neuron/synapse_types/synapse_types_exponential_impl.h

include ../Makefile.common
```

Python Interface – Why?



Python Interface – Class Definition

```
from spynnaker.pyNN.models.abstract_models.abstract_population_vertex import \
    AbstractPopulationVertex




class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
```

Python Interface – Class Definition

```
from spynnaker.pyNN.models.abstract_models.abstract_population_vertex import \
    AbstractPopulationVertex
from spynnaker.pyNN.models.abstract_models.abstract_model_components.\
    abstract_exp_population_vertex import AbstractExponentialPopulationVertex


class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
        AbstractExponentialPopulationVertex):
```

Python Interface – max atoms

```
class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
        AbstractExponentialPopulationVertex):

    _model_based_max_atoms_per_core = 255
```

Python Interface – initializer

```
class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
        AbstractExponentialPopulationVertex):

    _model_based_max_atoms_per_core = 255

    def __init__(
            self, n_neurons, machine_time_step, timescale_factor,
            spikes_per_second, ring_buffer_sigma, constraints=None, label=None,
```

Python Interface – initializer

```
class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
        AbstractExponentialPopulationVertex):

    _model_based_max_atoms_per_core = 255

    def __init__(
            self, n_neurons, machine_time_step, timescale_factor,
            spikes_per_second, ring_buffer_sigma, constraints=None, label=None,
            tau_syn_E=5.0, tau_syn_I=5.0,
```

Python Interface – initializer

```
class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
        AbstractExponentialPopulationVertex):

    _model_based_max_atoms_per_core = 255

    def __init__(
            self, n_neurons, machine_time_step, timescale_factor,
            spikes_per_second, ring_buffer_sigma, constraints=None, label=None,
            tau_syn_E=5.0, tau_syn_I=5.0,
            a=0.02, b=0.2, c=-65.0, d=2.0, i_offset=0, v_init=-70.0, u_init=-14.0):
```

Python Interface – initializer

```
class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
        AbstractExponentialPopulationVertex):

    _model_based_max_atoms_per_core = 255

    def __init__(
            self, n_neurons, machine_time_step, timescale_factor,
            spikes_per_second, ring_buffer_sigma, constraints=None, label=None,
            tau_syn_E=5.0, tau_syn_I=5.0,
            a=0.02, b=0.2, c=-65.0, d=2.0, i_offset=0, v_init=-70.0, u_init=-14.0):
        AbstractPopulationVertex.__init__(
            self, n_params=8,
```

Python Interface – initializer

```
class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
        AbstractExponentialPopulationVertex):

    _model_based_max_atoms_per_core = 255

    def __init__(
            self, n_neurons, machine_time_step, timescale_factor,
            spikes_per_second, ring_buffer_sigma, constraints=None, label=None,
            tau_syn_E=5.0, tau_syn_I=5.0,
            a=0.02, b=0.2, c=-65.0, d=2.0, i_offset=0, v_init=-70.0, u_init=-14.0):
        AbstractPopulationVertex.__init__(
            self, n_params=8, n_global_params=1,
```

Python Interface – initializer

```
class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
        AbstractExponentialPopulationVertex):

    _model_based_max_atoms_per_core = 255

    def __init__(
            self, n_neurons, machine_time_step, timescale_factor,
            spikes_per_second, ring_buffer_sigma, constraints=None, label=None,
            tau_syn_E=5.0, tau_syn_I=5.0,
            a=0.02, b=0.2, c=-65.0, d=2.0, i_offset=0, v_init=-70.0, u_init=-14.0):
        AbstractPopulationVertex.__init__(
            self, n_params=8, n_global_params=1, binary="IZK_curr_exp.aplx",
```

Python Interface – initializer

```
class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
        AbstractExponentialPopulationVertex):

    _model_based_max_atoms_per_core = 255

    def __init__(
            self, n_neurons, machine_time_step, timescale_factor,
            spikes_per_second, ring_buffer_sigma, constraints=None, label=None,
            tau_syn_E=5.0, tau_syn_I=5.0,
            a=0.02, b=0.2, c=-65.0, d=2.0, i_offset=0, v_init=-70.0, u_init=-14.0):
        AbstractPopulationVertex.__init__(
            self, n_params=8, n_global_params=1, binary="IZK_curr_exp.aplx",
            max_atoms_per_core=(IzhikevichCurrentExponentialPopulation.
                                _model_based_max_atoms_per_core),
```

Python Interface – initializer

```python
class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
        AbstractExponentialPopulationVertex):

    _model_based_max_atoms_per_core = 255

    def __init__(
            self, n_neurons, machine_time_step, timescale_factor,
            spikes_per_second, ring_buffer_sigma, constraints=None, label=None,
            tau_syn_E=5.0, tau_syn_I=5.0,
            a=0.02, b=0.2, c=-65.0, d=2.0, i_offset=0, v_init=-70.0, u_init=-14.0):
        AbstractPopulationVertex.__init__(
            self, n_params=8, n_global_params=1, binary="IZK_curr_exp.aplx",
            max_atoms_per_core=(IzhikevichCurrentExponentialPopulation.
                                _model_based_max_atoms_per_core),
            n_neurons=n_neurons,
            machine_time_step=machine_time_step,
            timescale_factor=timescale_factor,
            spikes_per_second=spikes_per_second,
            ring_buffer_sigma=ring_buffer_sigma,
            label=label, constraints=constraints)
```

Python Interface – initializer

```
class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
        AbstractExponentialPopulationVertex):

    _model_based_max_atoms_per_core = 255

    def __init__(
            self, n_neurons, machine_time_step, timescale_factor,
            spikes_per_second, ring_buffer_sigma, constraints=None, label=None,
            tau_syn_E=5.0, tau_syn_I=5.0,
            a=0.02, b=0.2, c=-65.0, d=2.0, i_offset=0, v_init=-70.0, u_init=-14.0):
        ...
        AbstractExponentialPopulationVertex.__init__(
            self, n_neurons=n_neurons, tau_syn_E=tau_syn_E,
            tau_syn_I=tau_syn_I, machine_time_step=machine_time_step)
```

Python Interface - Parameters

- Parameters can be:
  - Individual values
  - Array of values (one per neuron)
  - RandomDistribution
- Normalise Parameters
  - utility_calls.convert_param_to_numpy(
    param, n_neurons)

Python Interface – initializer

```
class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
        AbstractExponentialPopulationVertex):

    _model_based_max_atoms_per_core = 255

    def __init__(
            self, n_neurons, machine_time_step, timescale_factor,
            spikes_per_second, ring_buffer_sigma, constraints=None, label=None,
            tau_syn_E=5.0, tau_syn_I=5.0,
            a=0.02, b=0.2, c=-65.0, d=2.0, i_offset=0, v_init=-70.0, u_init=-14.0):
        ...
        ...
        self._a = utility_calls.convert_param_to_numpy(a, n_neurons)
        self._b = utility_calls.convert_param_to_numpy(b, n_neurons)
        self._c = utility_calls.convert_param_to_numpy(c, n_neurons)
        self._d = utility_calls.convert_param_to_numpy(d, n_neurons)
        self._i_offset = utility_calls.convert_param_to_numpy(i_offset, n_neurons)
        self._v_init = utility_calls.convert_param_to_numpy(v_init, n_neurons)
        self._u_init = utility_calls.convert_param_to_numpy(u_init, n_neurons)
```

Python Interface – state initializers

```
class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
        AbstractExponentialPopulationVertex):

    ...

    def initialize_v(self, v_init):
        self._v_init = utility_calls.convert_param_to_numpy(v_init, self.n_atoms)

    def initialize_u(self, u_init):
        self._u_init = utility_calls.convert_param_to_numpy(u_init, self.n_atoms)
```

Python Interface – properties

```
class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
        AbstractExponentialPopulationVertex):

    ...

    @property
    def a(self):
        return self._a

    @a.setter
    def a(self, a):
        self._a = utility_calls.convert_param_to_numpy(a, self.n_atoms)

    @property
    def b(self):
        return self._b

    @b.setter
    def b(self, b):
        self._b = utility_calls.convert_param_to_numpy(b, self.n_atoms)

    ...
```

Python Interface – model name

```
class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
        AbstractExponentialPopulationVertex):

    ...

    @property
    def model_name(self):
        return "IZK_curr_exp"
```

Python Interface – set max atoms

```
class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
        AbstractExponentialPopulationVertex):

    ...

    @staticmethod
    def set_model_max_atoms_per_core(new_value):
        IzhikevichCurrentExponentialPopulation.\
            _model_based_max_atoms_per_core = new_value
```

Python Interface – cpu usage

```
class IzhikevichCurrentExponentialPopulation(
        AbstractPopulationVertex
        AbstractExponentialPopulationVertex):

    ...

    def get_cpu_usage_for_atoms(self, vertex_slice, graph):
        return 782 * ((vertex_slice.hi_atom - vertex_slice.lo_atom) + 1)
```

Python Interface – parameters

```
class IzhikevichCurrentExponentialPopulation(
    ...
    def get_parameters(self):
        return [
            # REAL a
            NeuronParameter(self._a, DataType.S1615),
            # REAL b
            NeuronParameter(self._b, DataType.S1615),
            # REAL c
            NeuronParameter(self._c, DataType.S1615),
            # REAL d
            NeuronParameter(self._d, DataType.S1615),
            # REAL v
            NeuronParameter(self._v_init, DataType.S1615),
            # REAL u
            NeuronParameter(self._u_init, DataType.S1615),
            # REAL I_offset
            NeuronParameter(self._i_offset, DataType.S1615),
            # REAL this_h
            NeuronParameter(self._machine_time_step / 1000.0, DataType.S1615)
        ]
```

Python Interface – global params

```
class IzhikevichCurrentExponentialPopulation(
    ...
    def get_global_parameters(self):
        return [
            # REAL machine_timestep_ms
            NeuronParameter(self._machine_time_step / 1000.0, DataType.S1615)
        ]
```

Python Interface – abstract impl

```
class IzhikevichCurrentExponentialPopulation(
    ...
    def is_population_vertex(self):
        return True

    def is_exp_vertex(self):
        return True
```

New Model Template

- my_new_model
  - c_models
    - src
      - neuron
        - builds
          - my_model_curr_exp
            - Makefile
          - Makefile.common
        - models
          - neuron_model_my_model_curr_impl.c
          - neuron_model_my_model_curr_impl.h
        - Makefile.common
      - Makefile
  - examples
    - my_example.py
  - python_models
    - model_binaries
      - __init__.py
    - neural_models
      - __init__.py
      - my_model_curr_exp.py
    - __init__.py
  - setup.py

Using Your Model

```
import pyNN.spiNNaker as p
import python_models as new_models
```

Using Your Model

```python
import pyNN.spiNNaker as p
import python_models as new_models

my_model_pop = p.Population(
    1, new_models.MyModelCurrExp,
    {"my_parameter": 2.0,
     "i_offset": i_offset},
    label="my_model_pop")
```