Patch

Release 1.2.0

Robin De Schepper

Jan 25, 2020

CONTENTS:

CHAPTER

ONE

USAGE

Be aware that the interface is currently incomplete, this means that most parts are still "just" NEURON. I've only patched holes I frequently encounter myself when using the `h.Section`, `h.NetStim` and `h.NetCon` functions. Feel free to open an issue or fork this project and open a pull request for missing or broken parts of the interface.

1.1 Philosophy

Python interfaces should be Pythonic, this wrapper offers just that:

- Full Python objects: each wonky C-like NEURON object is wrapped in a full fledged Python object, easily handled and extended through inheritance.
- Duck typed interface: take a look at the magic methods I use and any object you create with those methods present will work just fine with Patch.
- Correct garbage collection, objects connected to eachother don't dissapear: Objects that rely on eachother store a reference to eachother. As is the basis for any sane object oriented interface.

1.2 Basic usage

Use it like you would use NEURON. The wrapper doesn't make any changes to the interface, it just patches up some of the more frequent and ridiculous gotchas.

Patch supplies a new HOC interpreter `p`, the `PythonHocInterpreter` which wraps the standard HOC interpreter `h` provided by NEURON. Any objects returned will either be `PythonHocObject`'s wrapping their corresponding NEURON object, or whatever NEURON returns.

When using just Patch the difference between NEURON and Patch objects is handled transparently, but if you wish to mix interpreters you can transform all Patch objects back to NEURON objects with `obj.__neuron__()` or the helper function `patch.transform`.

```
from patch import p, transform
import glia as g

section = p.Section()
point_process = g.insert(section, "AMPA")
stim = p.NetStim()
stim.start = 10
stim.number = 5
stim.interval = 10
```

(continues on next page)

(continued from previous page)

```
# And here comes the magic! This explicitly defined connection
# isn't immediatly garbage collected! What a crazy world we live in.
# Has science gone too far?
p.NetCon(stim, point_process)

# It's fully compatible using __neuron__
from neuron import h
nrn_section = h.Section()
nrn_section.connect(transform(section))
nrn_section.connect(section.__neuron__())
```

CHAPTER

TWO

SECTIONS

Sections are cilindrical representations of pieces of a cell. They have a length and a diameter. Sections are the main building block of a simulation in NEURON.

You can use the `.connect` method to connect *`Sections`* together.

Sections can be subdivided into *`Segments`* by specifying `nseg`, the simulator calculates the voltage for each segment, thereby affecting the spatial resolution of the simulation. The position of a segment is represented by its normalized position along the axis of the Segment. This means that a Segment at x=0.5 is in the middle of the Section. By default every section consists of 1 segment and the potential will be calculated for 3 points: At the start (0) and end (1) of the section, and in the middle of every segment (0.5). For 2 segments the simulator would calculate at 0, 0.333..., 0.666... and 1.

Patch

```
from patch import p
s = p.Section()
s.L = 40
s.diam = 0.4
s.nseg = 11

s2 = p.Section()
s.connect(s2)
```

NEURON

```
from neuron import h
s = h.Section()
s.L = 40
s.diam = 0.4
s.nseg = 11

s2 = h.Section()
s.connect(s2)
```

2.1 Retrieving segments

Sections can be called with an x to retrieve the segment at that x. The segments of a Section can also be iterated over.

Patch

```
s.nseg = 5
seg05 = s(0.5)
print(seg05)
for seg in s:
    print(seg)
```

NEURON

```
s.nseg = 5
seg05 = s(0.5)
print(seg05)
for seg in s:
    print(seg)
```

2.2 Recording

You can tell Patch to record the membrane potential of your Section at one or multiple locations by calling the `.record` function and giving it an `x`. If `x` is omitted `0.5` is used.

In NEURON you'd have to create a *`Vector`* and keep track of it somewhere and find a way to link it back to the Section it recorded, in Patch a section automatically stores its recording vectors in `section.recordings`.

Patch

```
s.record(x=1.0)
```

NEURON

```
v = h.Vector()
v.record(s(1.0))
all_recorders.append(v)
```

2.3 Position in space

With Patch it's very straightforward to define the 3D path of your Section through space. Call the `.add_3d` function with a 2D array containing the xyz data of your points. Optionally, you can pass another array of diameters.

Patch

```
s.add_3d([[0, 0, 0], [2, 2, 2]], diameters)
```

NEURON

```
s.push()
points = [[0, 0, 0], [2, 2, 2]]
for point, diameter in zip(points, diameters):
    h.pt3dadd(*point, diameter)
h.pop_section()
```

CHAPTER

THREE

MAGIC METHODS

3.1 __neuron__

Get the object's NEURON pointer

Whenever an object with this method present is sent to the NEURON HOC interpreter, the result of this method is passed instead. This allows Python methods to encapsulate NEURON pointers transparently

3.2 __netcon__

Get the object's NetCon pointer

Whenever an object with this method present is used in a *NetCon* call, the result of this method is passed instead. The connection is stored on the original object. This allows to simplify the calls to NetCon, or to add more elegant default behavior. For example inserting a connection on a section might connect it to a random segment and you'd be able to use `p.NetCon(section, synapse)`.

CHAPTER

FOUR

PATCH PACKAGE

4.1 patch.core module

patch.core.**transform**(*obj*)
Transforms an object to its NEURON representation, if the __neuron__ magic method is present.

patch.core.**transform_netcon**(*obj*)

patch.core.**transform_record**(*obj*)

4.2 patch.interpreter module

class patch.interpreter.**PythonHocInterpreter**
Bases: object

NetCon(*source*, *target*, **args*, ***kwargs*)

PointProcess(*factory*, *target*, **args*, ***kwargs*)
Creates a point process from a h.MyMechanism factory.

Parameters

- **factory** (*function*) – A point process method from the HocInterpreter.
- **target** (*objects.Segment*) – The Segment this point process has to be inserted into.

wrap(*factory*, *name*)

4.3 patch.objects module

class patch.objects.**NetCon**(*interpreter*, *ptr*)
Bases: *patch.objects.PythonHocObject*

class patch.objects.**NetStim**(**args*, ***kwargs*)
Bases: *patch.objects.PythonHocObject*, *patch.objects.connectable*

class patch.objects.**PointProcess**(**args*, ***kwargs*)
Bases: *patch.objects.PythonHocObject*, *patch.objects.connectable*

Wrapper for all point processes (membrane and synapse mechanisms). Use PythonHocInterpreter.PointProcess to construct these objects.

stimulate(***kwargs*)

class patch.objects.**PythonHocObject**(*interpreter*, *ptr*)
Bases: object

class patch.objects.**Section**(**args*, ***kwargs*)
Bases: *patch.objects.PythonHocObject*, *patch.objects.connectable*

add_3d(*points*, *diameters=None*)
Add a new 3D point to this section xyz data.

Parameters

- **points** – A 2D array of xyz points.
- **diameters** (*float or array*) – A scalar or array of diameters corresponding to the points. Default value is the section diameter.

connect(*target*, **args*, ***kwargs*)

connect_points(*target*, *x=0.5*)

insert(**args*, ***kwargs*)

record(*x=0.5*)

set_dimensions(*length*, *diameter*)

set_segments(*segments*)

wholetree()

class patch.objects.**Segment**(**args*, ***kwargs*)
Bases: *patch.objects.PythonHocObject*, *patch.objects.connectable*

class patch.objects.**Vector**(*interpreter*, *ptr*)
Bases: *patch.objects.PythonHocObject*

record(*target*)

class patch.objects.**connectable**
Bases: object

4.4 Module contents

patch.**connection**(*source*, *target*)

CHAPTER

FIVE

INSTALLATION

Patch can be installed using:

```
pip install nrn-patch
```

CHAPTER

SIX

KNOWN UNPATCHED HOLES

- When creating point processes the returned object is unwrapped. This can be resolved using Glia, or by using this syntax:

```
# In neuron
process = h.MyMechanismName(my_section(0.5), *args, **kwargs)
# In patch
point_process = p.PointProcess(p.MyMechanismName, my_section(0.5), *args, **kwargs)
```

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX