

Python and PyDSTool for Scientific Computation

Robert Clewley
Neuroscience Institute
Georgia State University

Python is ideal for scientific computing

- High level language, easy to learn
- Interpreted
- Object oriented
- Free and open source (portable)
- Easy to install on any platform
- Large, active, friendly user community
 - Drink the kool aid provided under your seat now!
- Easy to interface with C, Fortran, etc. (“glue”)
- Many 3rd party libraries to freely use
- Increasing support for parallel computation

Python widely adopted

- Already used by many government labs and industries
 - NASA, NIST, NCAR, NOAA, LLNL, LBL, Sandia, ...
 - Google, ILM, Walt Disney, Enthought, ...
- Actively developed language & tools
 - comp.lang.python is one of the biggest newsgroups
 - Mature and stable language (est. ~ 1992)
- Good time to add a modern skill to your CV!

Batteries are included

- “Introspection” at interactive prompt
- Debugger
- Help system built in: type `help(X)`
- Libraries for basic math, HTML/XML, I/O, OS extensions, ...

```
import math
math.sin(math.pi)
```

Essential libraries

- Numpy (multidimensional arrays, linear algebra, random numbers, FFT)
- Scipy (basic math and science functions, including stats & optimization)
- Matplotlib (robust 2D plotting)
- SymPy (symbolic computation, CAS)
- PyDSTool (dynamical systems, neural modeling of small networks, etc.)

Integrated Development Environments

- Free:
 - IPython very popular, simple
 - IDLE comes with, very simple
 - Wing IDE 101 is a full editor / debugger
 - Many others!
- \$\$:
 - Wing IDE Pro is excellent (cheap for academics)

Extensions with C/Fortran

- Weave (in-line C/C++ functions!)
- Cython (advanced, very fast)
- SWIG (older, but quite robust for many languages)
- F2PY (specifically for Fortran)

Python vs Matlab

- Main similarities for Python vs Matlab:
 - Interactive environment
 - Simple syntax
 - Dynamically typed (“duck typing”)
 - Automatic memory management (but Python’s is much more efficient)
 - Easy and fast to develop and test new ideas
 - Basic data types are essentially the same (strings, booleans, floats, lists)
 - Similar numeric performance
- Main differences for Python vs Matlab:
 - Indexing starts at 0 (like C)
 - No infix operators for linear algebra (use functions)
 - Whitespace (indentation) is used for code blocks
 - Much easier to interface with C/C++/Fortran
 - Functions can be defined anywhere (including inside other funcs)
 - Better container data types (esp. hash tables = “dictionaries”)
 - Proper object orientation (*everything* is an object and can be sub-classed)
 - Proper exception handling
 - It’s free, *dude*
 - A bit more fuss to install, set up, and manage (*but it’s good for you*)

Python vs Matlab

- Graphics:
 - Matplotlib 2D library mimics Matlab command interface
 - No standard 3D graphics or GUI builder: mlab (Mayavi2) and PyQwt are good
- Toolboxes:
 - So many now
 - http://www.scipy.org/Topical_Software is a good start
- Other differences and details:
 - http://www.scipy.org/NumPy_for_Matlab_Users

Resources

- python.org
- scipy.org
- matplotlib.sourceforge.net
- pydstool.sourceforge.net

- <http://metson.web.cern.ch/metson/PythonBeginners/PythonBeginners.pdf>
- <http://fperez.org/py4science/index.html>
- <http://www.physics.rutgers.edu/grad/509/python1.pdf>
- Many, many tutorials and examples online

- **Books:** (*Really?*) Langtangen's "Python Scripting for Computational Science"; Pilgrim's "Dive Into Python"

Installation options

- **Don't use version higher than Python 2.6 for now**
- **Windows**
 - Install each component from own .exe or .msi installers
 - Easy to select and configure, even uninstall is easy to change versions
 - Big all-in-one package: python(x,y)
 - restricted to PyQt and Eclipse IDE - *urgh* :(
- **Mac / linux** (probably don't want to use older pre-installed python)
 - Your favourite package manager (Fink for Mac (not Mac Ports), RPM, yum)
 - Must update path and .bashrc / .profile to point to correct installation
 - Some binaries available from http://www.scipy.org/Installing_SciPy/Linux
 - Python(x,y) on linux?
- **Any**
 - Compile everything from scratch using C and Fortran compilers
 - Good luck with that, especially for ATLAS/BLAS
 - Big all-in-one packages: ActiveState & Enthought
 - Enthought Python Distribution is free for academics
 - ActiveState's no longer bundles scientific software, uses PPM :(
 - Limited choice to preselect modules to install
 - May get a lot more than you wanted (300MB+)

Python fu to change your life

Dictionaries are fabulous

- Intuitive to use
 - A little bit like matlab cell arrays
- So many uses once you “get it”
- “Associative mapping” from keys to values
- Almost $O(1)$ lookup time (hash table)

```
d = { "a": 1, "b": 2, foo: bar }
d["a"]
d["c"] = 3
d = dict(a=1, b=2)
d = {}
d[(1,2)] = [8, 9]
print d.keys(), d.values(), len(d), d.items()
```

Classes and dynamic typing

- Dynamic typing with object orientation makes some calculations unbelievably easy to design and use
 - Dynamic typing means you don't hard-wire types of variables in advance
 - Names refer to any data type interchangeably
 - Only when used do the objects get tested to ensure suitability
 -
- E.g. cache for expensive re-used calculations (“memoize”)
 - make a class behave like a function but with memory

Memoize

```
class Memoize(object):
    """Memoize(fn) - an instance which acts like fn"""

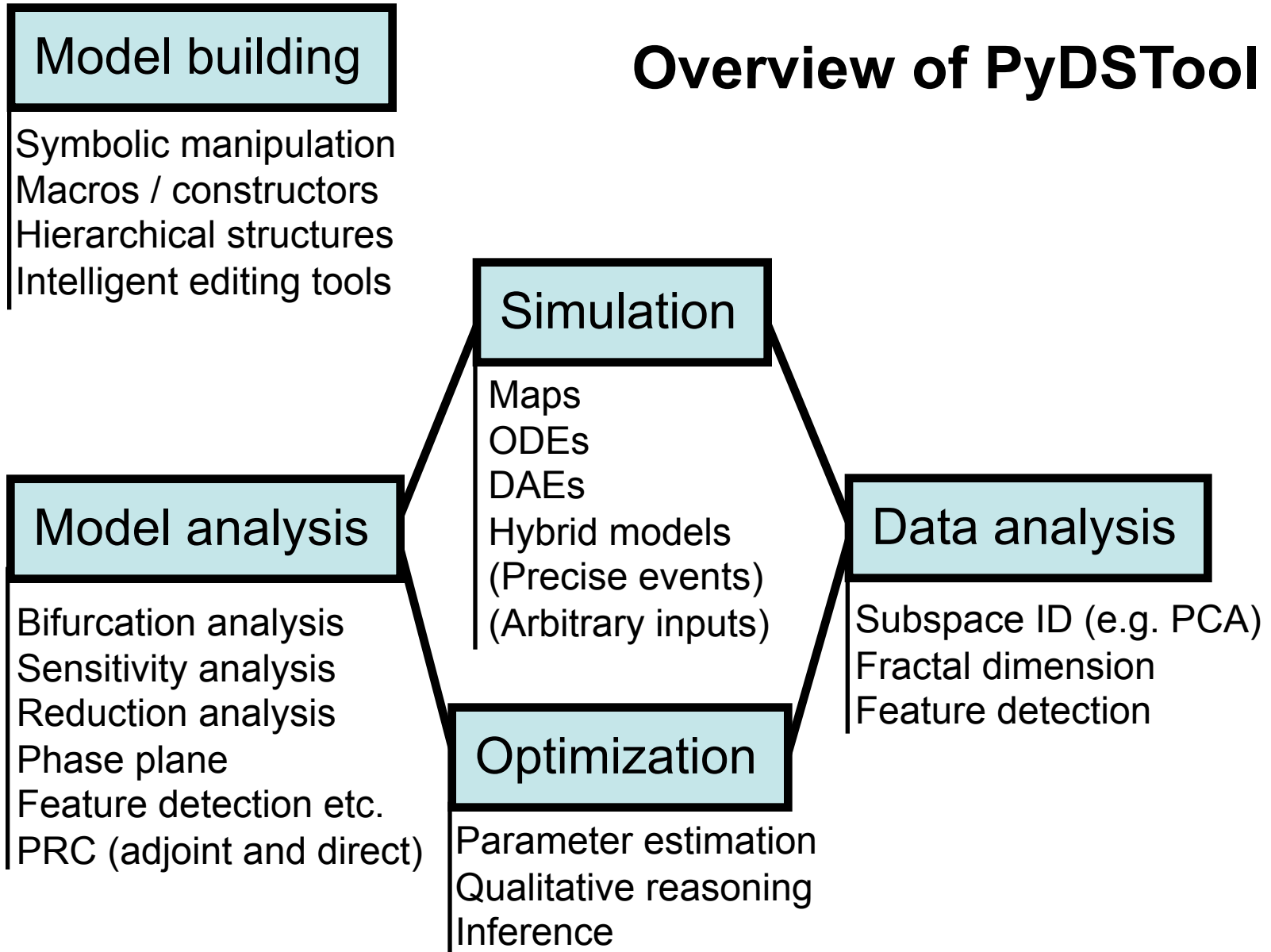
    def __init__(self, fn):    # "magic" initialization method
        self.fn = fn         # store reference to original function
        self.memo = {}       # empty dictionary

    def __call__(self, *args): # "magic" method to allow callability
        if not self.memo.has_key(args):
            # compute and store new value if not seen args before
            self.memo[args] = self.fn(*args)
        return self.memo[args]

def fib(n):
    if n < 2:
        return 1
    else:
        return fib(n-1) + fib(n-2)

fib = Memoize(fib)    # new behaviour is transparent to the end user!
```

Overview of PyDSTool



PyDSTool for ODEs, etc.

- Wiki at pydstool.sourceforge.net
 - Tutorial, installation instructions, documentation
 - Need gcc/gfortran (e.g., from MinGW) for fast ODE solvers
- Basic features:
 - Entirely scripted / command line (like Matlab)
 - Index-free variables equations
 - Can use indices for macro structure
 - Context heavy (e.g., description/tag/label attributes)
 - Equations easily defined by constructors or directly as strings
 - State/time events, auxiliary “helper” functions, external inputs, Jacobians, special points
 - Fast ODE/DAE solvers + AUTO interface built in
 - C RHS of your vector field is automatically produced!
 - Essentially as fast as native C code!

More features

- Pointset and Trajectory classes makes working with array data representing curves much more intuitive
- Supports hybrid dynamics (e.g., I&F, collisions, etc.)
- Basic phase plane tools
- Phase response curve tools (adjoint & direct methods)
- Modular network design tools (Cf. NEURON)
- Optimization / parameter estimation tools

Basic usage 1

```
from PyDSTool import *

pars = {'eps': 1e-2, 'a': 0.5}
icdict = {'x': pars['a'],
          'y': pars['a'] - pars['a']*pars['a']*pars['a']/3}
xstr = '(y - (x*x*x/3 - x))/eps'
ystr = 'a - x'

event_x_a = Events.makeZeroCrossEvent('x-a', 0,
                                       {'name': 'event_x_a',
                                        'eventtol': 1e-6,
                                        'term': False,
                                        'active': True},
                                       varnames=['x'], parnames=['a'])

DSargs = args(name='vanderpol') # struct-like data
DSargs.events = [event_x_a]
DSargs.pars = pars
DSargs.tdata = [0, 3]
DSargs.algparams = {'max_pts': 3000, 'init_step': 0.02}
DSargs.varspecs = {'x': xstr, 'y': ystr}
DSargs.ics = icdict
vdp = Vode_ODEsystem(DSargs)
```

Basic usage 2

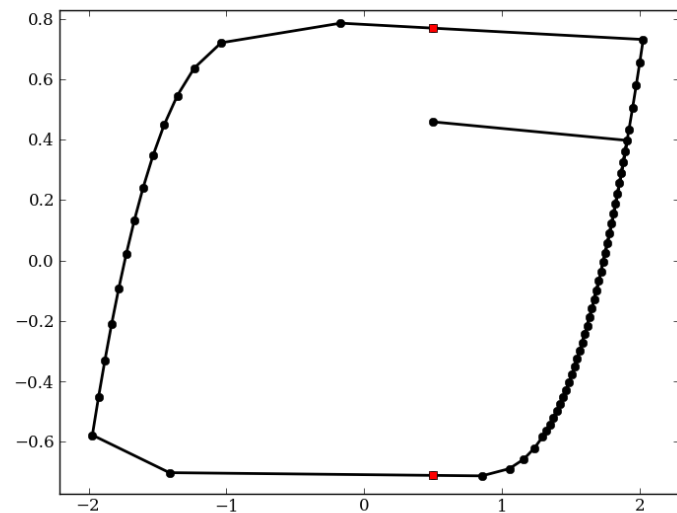
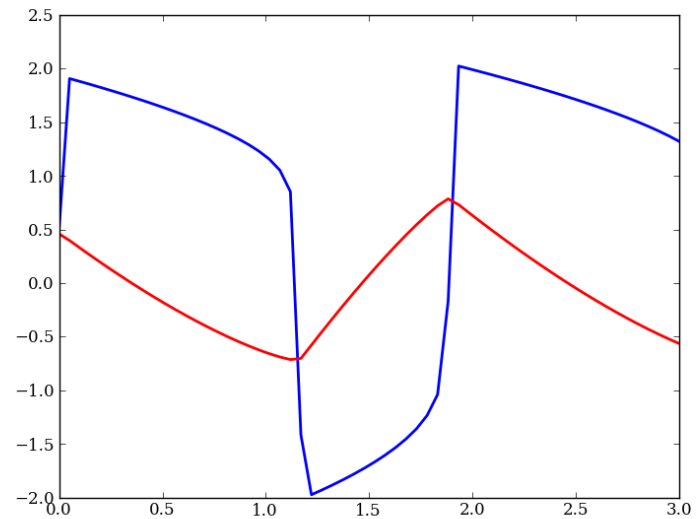
```
traj = vdp.compute('test_traj')  
pts = traj.sample()  
evs = traj.getEvents('event_x_a')
```

```
figure(1)  
plot(pts['t'], pts['x'], 'b',  
      linewidth=2)  
plot(pts['t'], pts['y'], 'r',  
      linewidth=2)
```

```
figure(2)  
plot(pts['x'], pts['y'], 'k-o',  
      linewidth=2)  
plot(evs['x'], evs['y'], 'rs')
```

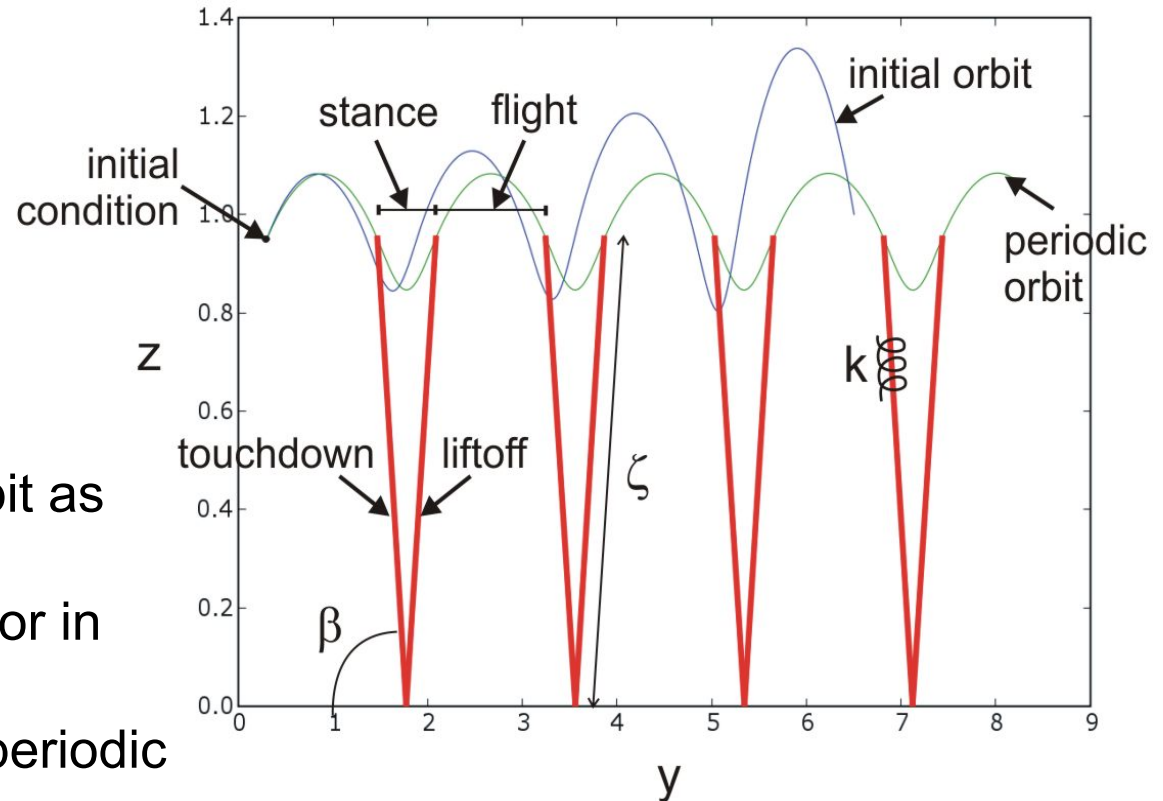
```
print traj(2.1)
```

```
show()
```



SLIP leg: analysis of a hybrid dynamical system

- 2D Spring-Loaded Inverted Pendulum leg model
- Two vector fields
 - Stance
 - Flight
- Two events
 - Touchdown
 - Liftoff
- Search for periodic orbit as a function of β
- Embed call to Generator in residual function
- Determine stability of periodic orbit from return map



Optimization residual function

```
def f( $\beta$ ): # pseudo-code!  
    set SLIP parameter value to  $\beta$   
    compute new orbit from touchdown to liftoff  
    (stop at liftoff event)  
    liftoff angle  $\delta = \text{asin}(\text{liftoff\_height})$   
    return residual =  $(\beta - \delta)^2$ 
```

```
 $\beta_{\text{opt}} = \text{fminbound}(f, \beta_{\text{lo}}, \beta_{\text{hi}}, \text{tol}=10^{-3})$   
ic_pdc = copy(ic) # for ydot, zdot  
ic_pdc['z'] = sin( $\beta_{\text{opt}}$ )  
ic_pdc['y'] = cos( $\beta_{\text{opt}}$ )
```

Liftoff-to-liftoff discrete mapping

```
SLIP = makeSLIP2D(pars, stop_at_liftoff=True)

def P(ic): # the mapping, embedding calls to the model
    SLIP.compute(trajname='pdc', force=True,
                 ics=ic, tdata=[0,12], verboselevel=0)
    # extract first point from pointset using [0] reference
    res = SLIP.getTrajEvents('pdc', 'liftoff')[0]
    res['incontact'] = 0 # make eligible for new i.c.
    return res

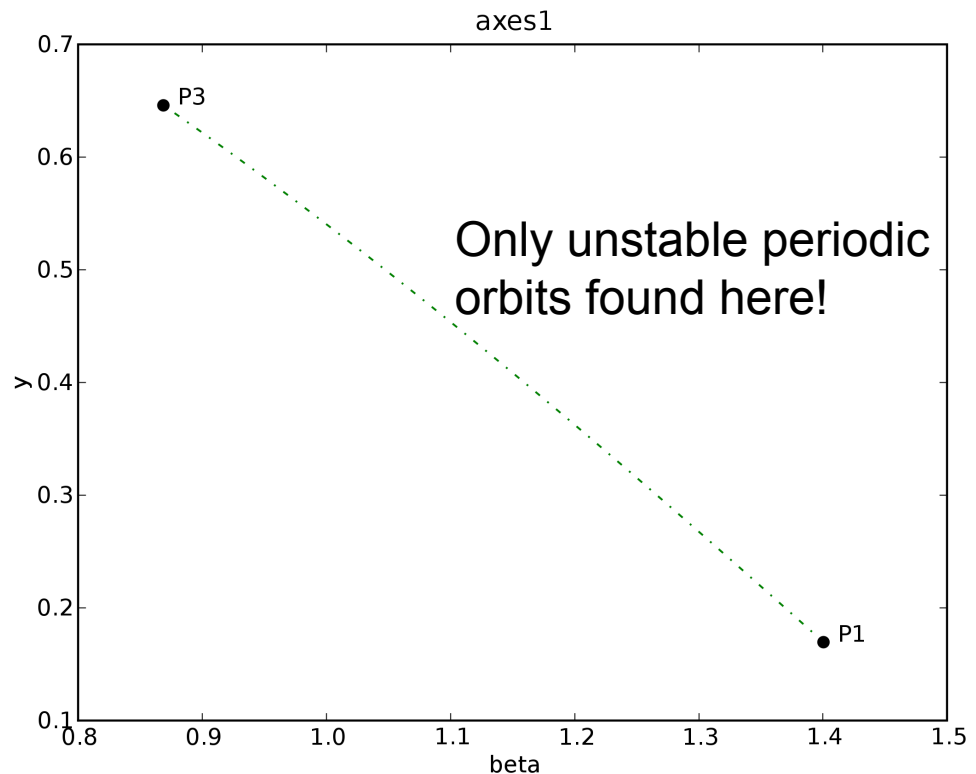
# numeric differentiation using Ridder's method
DP = diff(P, ic_pdc, axes=['z', 'ydot', 'zdot'],
          vars=['z', 'ydot', 'zdot'], eps=1e-2)

# DP is a 3 x 3 float array
eigvals(DP) => 0.0001126, 0.9996, 1.1760 so orbit is unstable
```

PyCont looks for stable regime

```
pc=ContClass(map) # 4D return map embedded in MapSystem
pc.newCurve(args(name='FPu', type='FP-C', freepars=['beta'],
    StepSize = 2e-2, MaxStepSize = 3e-2, MaxNumPoints = 50,
    LocBifPoints = 'all', SaveEigen = True,
    verbosity = 3,
    TestTol = 4e-6, FuncTol = 4e-6, VarTol = 4e-6
    ))
```

```
pc['FPu'].forward()
pc['FPu'].backward()
pc['FPu'].forward()
pc['FPu'].cleanLabels()
pc.display(stability=True)
```



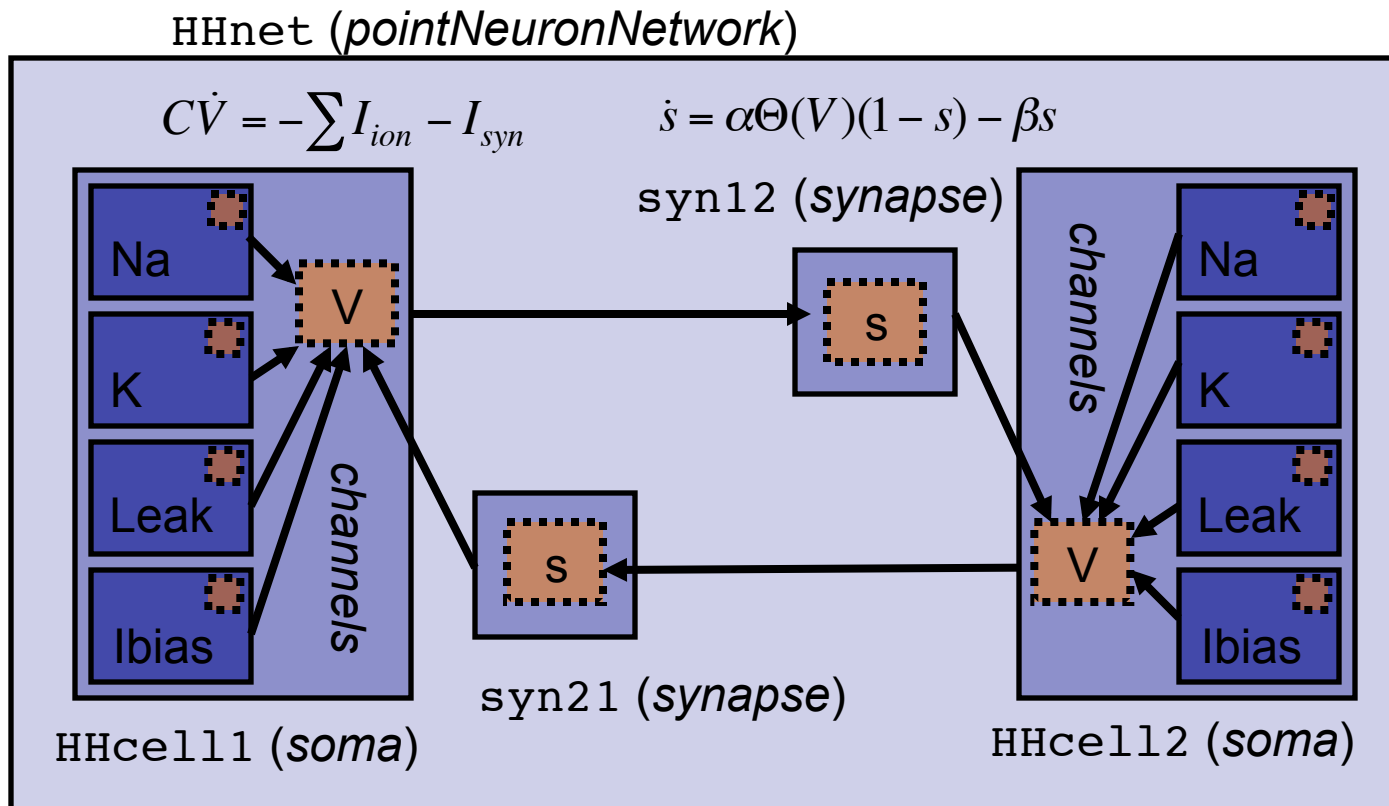
Solution pointset stores extra info

```
>>> pc['Fpu'].sol[5]

beta: 1.34271349996
y: 0.226110418619
ydot: 0.440519192137
z: 0.974101677751
zdot: 0.300016228842
Labels: FP ({'data': Arguments:
  evals : [ 0.00000000e+00+0.j 0.00000000e+00+0.j -3.88419548e-04+0.j
            1.35473530e+00+0.j]
  ds : 0.03
  evects : [[ 1.00000000e+00 0.00000000e+00 5.13300867e-04 -2.31602889e-01]
             [ 0.00000000e+00 0.00000000e+00 1.90657085e-07 -8.46134811e-01]
             [ 0.00000000e+00 0.00000000e+00 -3.38018149e-04 1.52514759e-01]
             [ 0.00000000e+00 1.00000000e+00 9.99999811e-01 4.55143089e-01]]
  V : {'y': 0.285331616137, 'beta': -0.300400357132, 'z': -0.0939480287614,
        'ydot': 0.90527303183, 'zdot': 1.90348269714e-07},
  'domain': 'inside',
  'stab': 'N'})

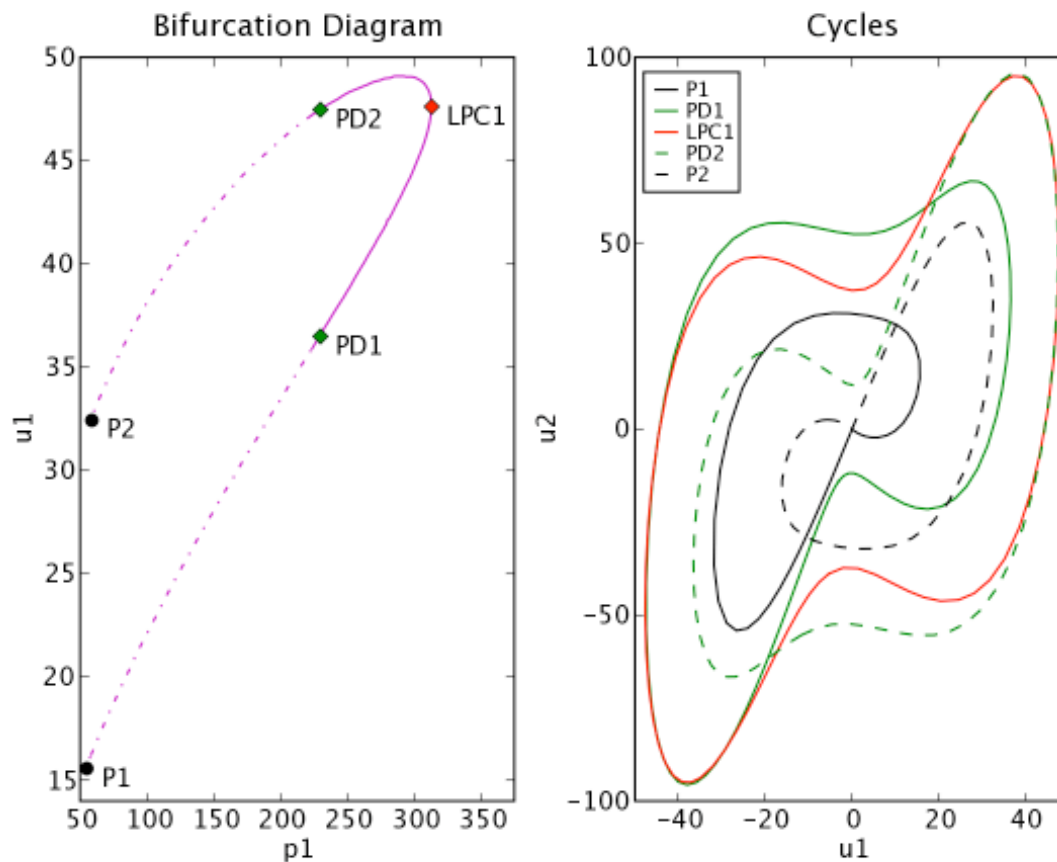
>>> pc['Fpu'].sol[5].labels['FP'] # direct access to any label
```

Simple HH interneuron network



Interface to AUTO is the nicest you'll see

(AUTO now officially uses part of PyDSTool code in its user interface)



Acknowledgments

- NSF Cyber-enabled Discovery and Innovation award:
Emerging Models and Technology for Computation /
BSSE #0829742
- Erik Sherwood
- Drew Lamar
- John Guckenheimer