Xolotl

A fast and flexible neuronal simulator

Alec Hoyland

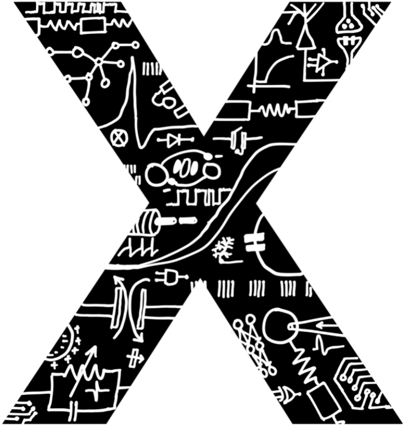
Center for Systems Neuroscience

March 22, 2019

Structure of Talk

"What" more than "Why and How"

1. What is xolotl?
2. Features
3. Demonstrations
   1. My first neuron
   2. My first network
   3. Demos & interactive demos

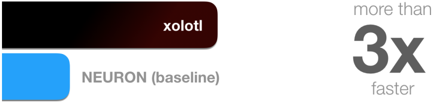


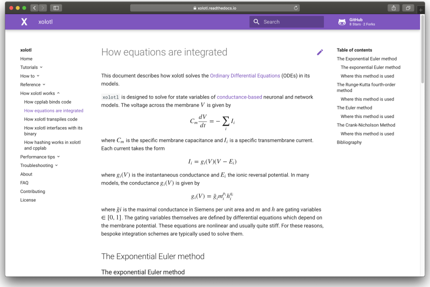
Contact me at: ahoyland@bu.edu

Design Principles

Xolotl should be

- fast
- easy-to-use
- well-documented
- hackable and extensible
- auditable




xolotl.readthedocs.io
X xolotl
Search
GitHub
8 Stars · 2 Forks
xolotl
Home
Tutorials
How to
Reference
How xolotl works
How cpplab binds code
How equations are integrated
How xolotl transpiles code
How xolotl interfaces with its binary
How hashing works in xolotl and cpplab
Performance tips
Troubleshooting
About
FAQ
Contributing
License
Table of contents
The Exponential Euler method
The exponential Euler method
Where this method is used
The Runge-Kutta fourth-order method
Where this method is used
The Euler method
Where this method is used
The Crank-Nicholson Method
Where this method is used
Bibliography

How equations are integrated

This document describes how xolotl solves the Ordinary Differential Equations (ODEs) in its models.

`xolotl` is designed to solve for state variables of conductance-based neuronal and network models. The voltage across the membrane V is given by

$$C_m \frac{dV}{dt} = - \sum_i I_i$$

where C_m is the specific membrane capacitance and I_i is a specific transmembrane current. Each current takes the form

$$I_i = g_i(V)(V - E_i)$$

where $g_i(V)$ is the instantaneous conductance and E_i the ionic reversal potential. In many models, the conductance $g_i(V)$ is given by

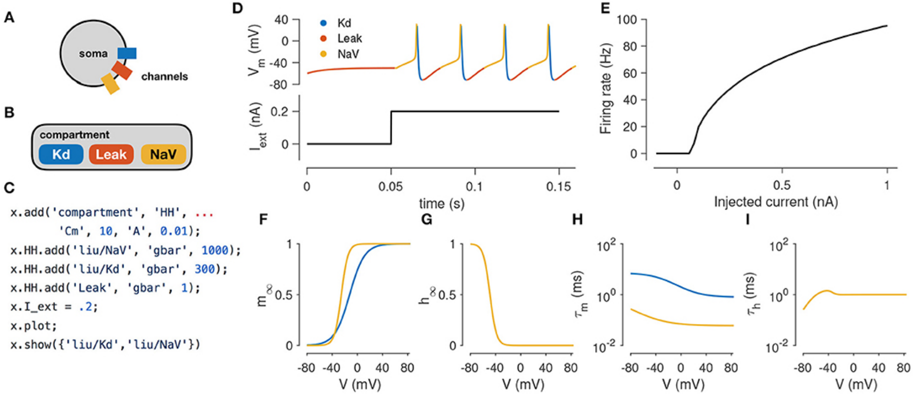
$$g_i(V) = \bar{g}_i m_i^{p_i} h_i^{q_i}$$

where $\bar{g}i$ is the maximal conductance in Siemens per unit area and m and h are gating variables $\in [0, 1]$. The gating variables themselves are defined by differential equations which depend on the membrane potential. These equations are nonlinear and usually quite stiff. For these reasons, bespoke integration schemes are typically used to solve them.

The Exponential Euler method

The exponential Euler method

How xolotl works



```
x.add('compartment', 'HH', ...
      'Cm', 10, 'A', 0.01);
x.HH.add('liu/NaV', 'gbar', 1000);
x.HH.add('liu/Kd', 'gbar', 300);
x.HH.add('Leak', 'gbar', 1);
x.I_ext = .2;
x.plot;
x.show({'liu/Kd','liu/NaV'})
```

Figure: Model of A & B represented in code C which produces D-I.

Anatomy of a model

Types of Components:

- Compartments
  - Mechanisms
  - Conductances
    - Mechanisms
  - Synapses
    - Mechanisms

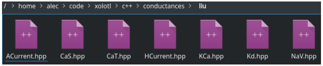


Figure: 100+ components are a searchable, indexed feature of the language.

Code to generate an HH model with constant injected current:

```
x = xolotl;
x.add('compartment', 'HH', 'Cm', 10, 'A', 0.01);
x.HH.add('liu/NaV', 'gbar', 1000);
x.HH.add('liu/Kd', 'gbar', 300);
x.HH.add('Leak', 'gbar', 1);
x.I_ext = 0.2;


------------------------------------------------
% How xolotl prints in the console
------------------------------------------------


>> x
xolotl object with
---------------------
+ HH
  > Kd (g=300, E=-80)
  > Leak (g=1, E=-55)
  > NaV (g=1000, E=30)
---------------------
```

Cool features

- `puppeteer`: real-time parameter optimization
- `xgrid`: parallel simulation across a distributed network
- `xfit`: parameter optimization using particle swarm and genetic algorithms
- `xtools`: spike counting and data analysis
- model hashing and snapshotting
- control over input and output (clamping, full state matrix)
- automatic component generation from MATLAB
- hyperlinking and tab-completion in the console
- multiple solvers, look-up table caching

Coming Soon

- multi-threading of a single simulation
- server-side compilation / stand-alone integration
- (multi-compartment) server-side GPU computation of Hines matrices
- new compartment types (including low-dimensional models)
- universal support for Runge-Kutta integration schemes
- adaptive time-step solvers (quadrature)
- robust front-end unit support
- Julia front-end (compatible with Python, etc.)

Real-time manipulation

Real-time parameter manipulation. Any numerical xolotl property can be manipulated.

xfit: Parameter optimization

Optimized for:

1. Slow-wave troughs at -70 mV.
2. Slow-wave peaks at -40 mV.
3. Spike downswing ends above slow wave trough.
4. Burst frequency of 0.5 Hz.
5. Duty cycle of 0.3.

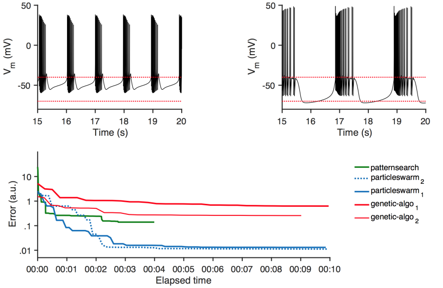


Figure: Fit of 8-conductance model (left to right). PSO #2 shown.

Installing

Acquiring the MATLAB toolbox

1. Go to https://github.com/sg-s/xolotl/releases/
2. From the `22-March-2019` release, download `xolotl.mltbox`
3. Find the file in `Downloads` and drag it onto your `MATLAB` workspace
   This will install `xolotl`

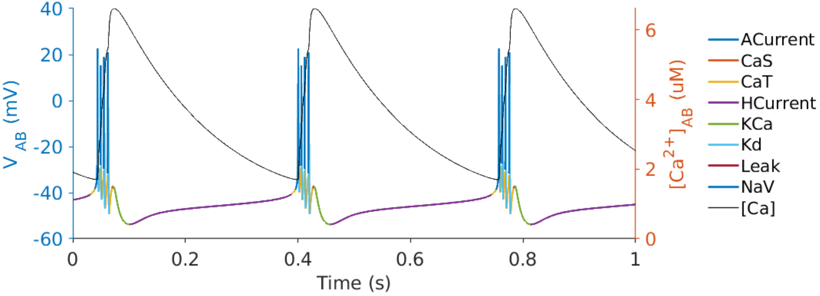
https://xolotl.readthedocs.io/en/master/tutorials/start-here/

Run the following commands in MATLAB. You should see this plot.

```
% setup the C++ compiler
mex -setup c++
mex -setup c
% click the link for the MinGW64 Compiler (C++)

% rebuild the component cache
xolotl.rebuildCache

% test to make sure everything is correct
xolotl.go_to_examples
demo_bursting_neuron
```



https://xolotl.readthedocs.io/en/master/tutorials/built-in-demos/

Demonstrations

Your first neuron

https://xolotl.readthedocs.io/en/master/tutorials/first-neuron/

Your first network

https://xolotl.readthedocs.io/en/master/tutorials/first-network/

All demos

https://xolotl.readthedocs.io/en/master/tutorials/built-in-demos/