Frontiers in NeuroInformatics

Methods
November 28, 2013

NeuronUnit: Collaborative Validation Testing of Neurophysiological Models

Richard C. Gerkin [1,*] **and Cyrus Omar** [2]

[1] *School of Life Sciences, Arizona State University, Tempe, AZ, USA*
[2] *Dept. of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA*

Correspondence*:
Rick Gerkin
ISTB-1 476, rgerkin@asu.edu

Python in Neuroscience II

ABSTRACT

Rigorously validating a scientific model's explanatory power requires comparing its predictions against empirical data on an ongoing basis. However, model validation remains an informal and incomplete process, especially in fields like neuroscience where the available data is growing rapidly. This has made it difficult to develop compare competing models, to determine the state-of-the-art, and to precisely identify open modeling problems.

Software engineers validate software by writing simple executable tests, called "unit tests". Suites of such unit tests can collectively provide evidence for its validity and correctness. Drawing inspiration from this practice, we develop the *SciUnit* framework for developing suites of "model validation tests" – executable functions, here written in Python, that compare model predictions against a single empirical observation to produce a score indicating agreement between the model and the data. Suites of such validation tests are used as a summary of modeling progress within a scientific community. As a first application of this framework, we describe *NeuronUnit*, a library that supports validation testing for neurophysiology and integrates with several existing neuroinformatics resources.

Keywords: Neuroinformatics Simulation Electrophysiology Software Modeling

1 INTRODUCTION

Neuroscientists construct quantitative models to coherently explain observations of neurons, circuits, brain regions and behavior. These models can be characterized by their *scope*: the set of observable quantities that the model can generate predictions about, and by their *validity*: the extent to which these predictions agree with empirical observations of those quantities.

Today, scientists contribute a new model to the research community by submitting a paper describing how the model works, along with selected figures demonstrating its scope and validity and comparing its validity to other models with the same scope. Reviewers are then responsible for discovering relevant data and competing models that the paper did not adequately consider, drawing on their knowledge of prior publications. However, in many areas, the number of publications being generated every year can overwhelm even the most conscientious scientists (**Jinha**, 2010).

Unfortunately, there are few alternatives to a comprehensive literature review available when scientists need to answer questions like these:

- Which models are capable of predicting the quantities I am interested in?
- Which metrics should be used to evaluate the goodness-of-fit between these models and data?
- How well do these models perform, as judged to these metrics, given currently available data?
- What other quantities can and can't these models predict?
- What observations are not adequately explained by any available model?

Professional software developers face similar issues. They must understand the scope of each component of a complex software project and validate it by measuring how well each component achieves its specified input/output behavior. But software developers do not validate components by simply choosing a few interesting inputs and presenting the outputs to reviewers. Rather, they typically follow a *test-driven development* methodology by creating a suite of executable *unit tests* that serve to specify each component's scope and validate its implementation as it is being developed and modified (**Beck**, 2003). Each test individually checks that a small portion of the program meets a single correctness criterion. For example, a unit test might verify that one function within the program correctly handles malformed inputs. Collectively, the test results serve as a summary of the validity of the project as it progresses through its development cycle. Developers can determine which features are unimplemented or buggy by examining the set of failed tests, and progress can be measured in terms of how many tests the program passes over time. This methodology is widely adopted in practice (**Beck**, 2003).

Test-driven methodologies have started to see success in neuroscience as well. Modeling competitions in neuroscience, for example, are typically organized around a collection of simple validation criteria, implemented as executable tests. These competitions continue to drive important advances and improve scientists' understanding of the relative merits of different models. For example, the quantitative single neuron modeling competition (QSNMC) (**Jolivet et al.**, 2008) investigates the complexity-accuracy tradeoff among reduced models of excitable membranes; the "Hopfield" challenge (**Hopfield and Brody**, 2000) tested techniques for generating neuronal network form given its function; the Neural Prediction Challenge sought the best stimulus reconstructions, given neuronal activity (*http://neuralprediction.berkeley.edu*); the Diadem challenge is advancing the art of neurite reconstruction (*http://www.diademchallenge.org*); and examples from other subfields of biology abound (*http://www.the-dream-project.org*).

Each of these examples has leveraged *ad hoc* infrastructure to support test generation. While the specific criteria used to evaluate models varies widely between disciplines in neuroscience, the underlying test-driven methodology has many common features that could be implemented once. Recognizing this, we developed a discipline-agnostic framework for developing scientific validation test suites called *SciUnit* (*http://www.sciunit.org*). Here we describe *NeuronUnit*, which builds upon *SciUnit*, allowing neuroscientists to build *SciUnit* tests that validate neurophysiology models against electrophysiological data. We provide a concrete example pipeline, showing how models described using NeuroML and provided freely by the *Open Source Brain Project* (OSB, **Gleeson et al.** (2012), *http://www.opensourcebrain.org*) can be tested in fully automated fashion using published, curated data available through the *NeuroElectro Project* (Neuroelectro, **Tripathy et al.** (2012), *http://neuroelectro.org*), leveraging facilities from the *NeuroTools* library (*http://neuralensemble.org/NeuroTools*) to extract relevant features of model output. This is summarized in Figure 1, which shows the relationships between the layers described here.

2 VALIDATION TESTING WITH *SCIUNIT*

2.1 EXAMPLE: THE QUANTITATIVE SINGLE NEURON MODELING COMPETITION

We first illustrate the form of a generic example *SciUnit* test suite that could be used in neurophysiology. Suppose we have collected data from an experiment where current stimuli (measured in pA) are delivered to neurons in some brain region, while the somatic membrane potential of each stimulated cell (in mV) is

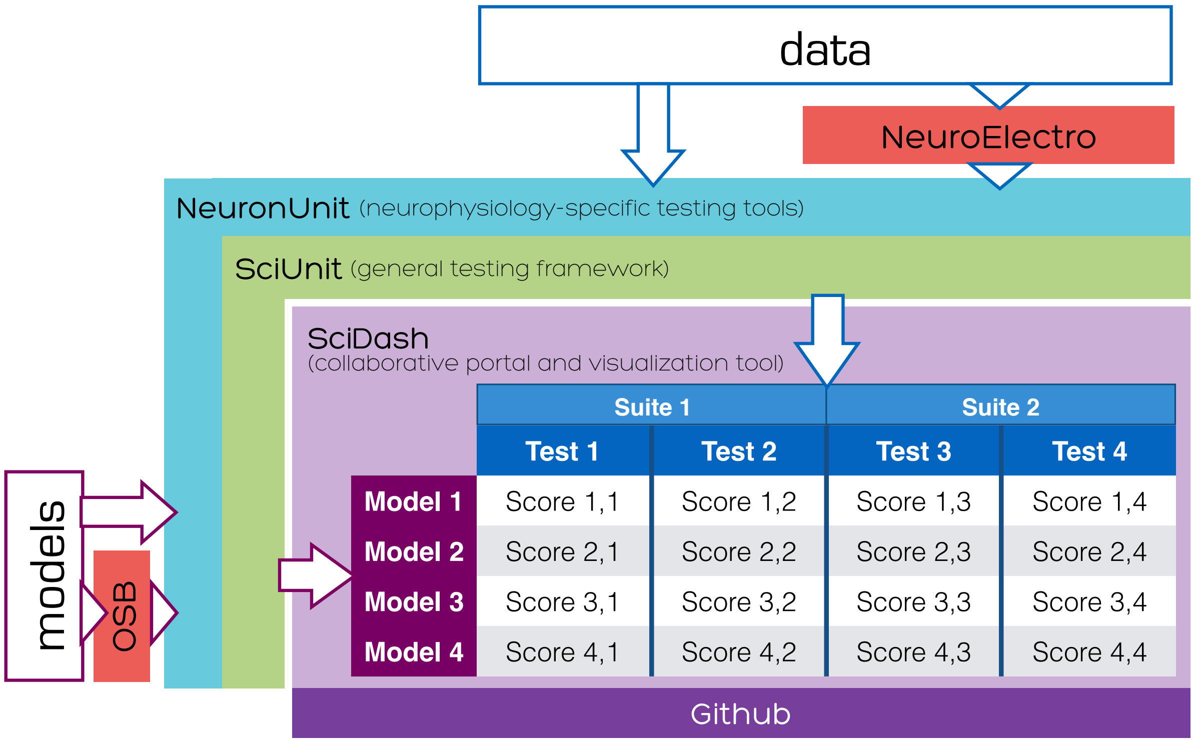


Figure 1. NeuronUnit overview. NeuronUnit is set of testing tools built upon the discipline-agnostic SciUnit framework. NeuronUnit can in principle test arbitrary neurophysiology models using arbitrary data but we provide here an example using models described in NeuroML as part of the *Open Source Brain Project* (OSB, (**Gleeson et al.**, 2012), *http://www.opensourcebrain.org*), and single neuron electrophysiology data available as part of the NeuroElectro Project (Neuroelectro, (**Tripathy et al.**, 2012), *http://neuroelectro.org*). Records of test results for various model/test combinations are accessible via SciDash, which indexes GitHub repositories of these records, models, and tests so they can be searched and filtered by the community.

recorded and stored. A model claiming to capture this cell type's membrane potential dynamics must be able to accurately predict a variety of features observed in these data.

One simple validation test would ask candidate models to predict the number of action potentials (a.k.a. spikes) generated in response to a stimulus (e.g. white noise), and compare these *spike count* predictions to the distribution observed in repeated experimental trials using the same stimulus. For data of this type, goodness-of-fit can be measured by first calculating a p-value from a chi-squared statistic for each prediction and then combining these p-values using Fisher's method (**Fisher**, 1925).

Alongside this *spike count test*, we might also specify a number of other tests capturing different features of the data to produce a more comprehensive suite. For data of this sort, the QSNMC defined 17 other validation criteria in addition to one based on the overall spike count, capturing features like spike latencies (SL), mean subthreshold voltage (SV), interspike intervals (ISI) and interspike minima (ISM) that can be extracted from the data (**Jolivet et al.**, 2008). They then defined a combined metric favoring models that broadly succeeded at meeting these criteria, to produce an overall ranking. Such combined criteria are simply validation tests that invoke other tests to produce a result.

2.2 IMPLEMENTING A VALIDATION TEST IN *SCIUNIT*

Fig. 2 shows how a scientist can implement spike count tests such as the one described above using *SciUnit*. A *SciUnit* validation test is an instance (i.e. an object) of a Python class implementing the `sciunit.Test` interface (cf. line 1). Here, we show a class `SpikeCountTest` taking three *parameters* in its constructor (constructors are named `__init__` in Python, lines 9-10). The meaning of each

```
 1 class SpikeCountTest(sciunit.Test):
 2   """Tests spike counts produced in response to several current stimuli against observed means and
        standard deviations.
 3
 4   goodness of fit metric: Computes p-values based on a chi-squared test statistic, and pools them
        using Fisher's method.
 5   parameters:
 6     inputs: list of numpy arrays containing input currents (nA)
 7     means, stds: lists of observed means and standard deviations, one per input
 8   """
 9   def __init__(self, inputs, means, stds):
10     self.inputs, self.means, self.stds = inputs, means, stds
11
12   required_capabilities = [SpikeCountFromCurrent]
13
14   def _judge(self, model):
15     inputs, means, stds = self.inputs, self.means, self.stds
16     n = len(inputs)
17     counts = numpy.empty((n,))
18     for i in xrange(n):
19       counts[i] = model.spike_count_from_current(inputs[i])
20     chisquared = sum((counts-means)**2 / means) # An array of chi-squared values.
21     p = scipy.stats.chi2.cdf(chisquared,n-1) # An array of p-values.
22     pooled_p = sciunit.utils.fisherp(p_array) # A pooled p-value.
23     return sciunit.PValue(pooled_p, related_data={
24       "inputs": inputs, "counts": counts, "obs_means": means, "obs_stds": stds
25     })
```

Figure 2. An example single neuron spike count test class implemented using *SciUnit*. Because this implementation contains logic common to many different systems, *NeuronUnit* was developed to provide a simpler means to deliver it (see Sec. 3.1).

parameter along with a description of the goodness-of-fit metric used by the test is documented on lines 4-7. To create a *particular* spike count test, we instantiate this class with particular experimental observations. For example, given observations from hippocampal CA1 cells (not shown), we can instantiate a test as follows:

use iPython style In[0] notation?

```
CA1_sc_test = SpikeCountTest(CA1_inputs, CA1_means, CA1_stds)
```

We emphasize the crucial distinction between the *class* `SpikeCountTest`, which defines a *parameterized family* of validation tests, and the particular *instance* `CA1_sc_test`, which is an individual validation test because the necessary parameters, derived from data, have been provided. As we will describe below, we expect communities to build repositories of such families capturing the criteria used in their subfields of neuroscience so that test generation for a particular system of interest will often require simply instantiating a previously-developed family with particular experimental parameters and data. For single-neuron test families like `SpikeCountTest`, we have developed such a library, called *NeuronUnit* (*http://github.com/scidash/neuronunit*) (Sec. 3.1).

Classes that implement the `sciunit.Test` interface must contain a `_judge` method that receives a candidate *model* as input and produces a *score* as output. To specify the interface between the test and the model (that is, to specify an appropriate scope), the test author provides a list of *capabilities* in the `required_capabilities` attribute, seen on line 12 of Fig. 2. Capabilities are simply collections of methods that a test will need to invoke in order to receive relevant data, and are analogous to *interfaces* in e.g. Java (*http://docs.oracle.com/javase/tutorial/java/concepts/interface.html*). In Python, capabilities are written as classes with unimplemented members. The capability required by the test in Fig. 2 is shown in Fig. 3. In *SciUnit*, classes defining capabilities are tagged as such by inheriting from `sciunit.Capability`. The test in Figure 2 uses this capability on line 19 to produce a spike count prediction for each input current.

```python
class SpikeCountFromCurrent(sciunit.Capability):
  def spike_count_from_current(self, input):
    """Takes a numpy array containing current stimulus (in nA) and
    produces an integer spike count. Can be called multiple times."""
    raise NotImplementedError("Model does not implement capability.")
```

Figure 3. An example capability specifying a single required method (used by the test in Figure 2).

```python
class TrainSpikeCountFromCurrent(sciunit.Capability):
  def train_with_currents(self, currents, counts):
    """Takes a list of numpy arrays containing current stimulus (in nA) and
    observed spike counts. Model parameters should be adjusted based on this
    training data."""
    raise NotImplementedError("Model does not implement capability.")
```

Figure 4. Another capability specifying a training protocol (not used by the test in Figure 2).

The remainder of the `_judge` method implements the goodness-of-fit metric described above, returning an instance of `sciunit.PValue`, a subclass of `sciunit.Score` that is included with *SciUnit*. In addition to the p-value itself, the returned score object also contains metadata, via the `related_data` parameter, for scientists who may wish to examine the result in more detail later. In this case we save the input currents, the model outputs and the observed means and standard deviations (line 24).

2.3 MODELS

Capabilities are *implemented* by models. In *SciUnit*, models are instances of Python classes that inherit from `sciunit.Model`. Like tests, the class itself represents a family of models, parameterized by the arguments of the constructor. A particular model is an instance of such a class.

Figure 5 shows how to write a simple family of models, `LinearModel`, that implement the capability in Fig. 3 as well as another capability shown in Fig. 4, which we will discuss below. Models in this family generate a spike count by applying a linear transformation to the mean of the provided input current. The family is parameterized by the scale factor and the offset of the transformation, both scalars. To create a *particular* linear model, a modeler can provide particular parameter values, just as with test families:

```python
CA1_linear_model_heuristic = LinearModel(3.0, 1.0)
```

Here, the parameters to the model were picked by the modeler heuristically, or based on externally-available knowledge. An alternative test design would add a training phase where these parameters were fit to data using the capability shown in Fig. 4. This test could thus only be used for those models for which parameters can be adjusted without human involvement. Whether to build a training phase into the test protocol is a choice left to each test development community.

Fig. 2 does not include a training phase. If training data is externally available, models that nevertheless do implement a training capability (like `LinearModel`) can simply be trained explicitly by calling the capability method just like any other Python method:

```python
CA1_linear_model_fit = LinearModel()
CA1_linear_model_fit.train_with_currents(CA1_training_in, CA1_training_out)
```

2.4 EXECUTING TESTS

A test is executed against a model using the `judge` method:

```python
score = CA1_sc_test.judge(CA1_linear_model_heuristic)
```

```python
class LinearModel(sciunit.Model, SpikeCountFromCurrent,
    TrainSpikeCountFromCurrent):
  def __init__(self, scale=None, offset=None):
    self.scale, self.offset = scale, offset

  def spike_count_from_current(self, input):
    return int(self.scale*numpy.mean(input) + self.offset)

  def train_with_currents(self, currents, counts):
    means = [numpy.mean(c) for c in currents]
    [self.offset, self.scale] = numpy.polyfit(means, counts, deg=1)
```

Figure 5. A model that returns a spike count by applying a linear transformation to the mean input current. The parameters can be provided manually or learned from data provided by a test or user (see text).

This method proceeds by first checking that the provided model implements all required capabilities. It then calls the test's `_judge` method to produce a score. A reference to the test and model are added to the score for convenience (accessible via the `test` and `model` attributes, respectively), before it is returned.

2.5 TEST SUITES AND SCORE MATRICES

A collection of tests intended to be run on the same model can be put together to form a test suite. The following is a test suite that could be used for a simplified version of the QSNMC, as described above:

```python
CA1_suite = sciunit.TestSuite([CA1_sc_test, CA1_sl_test, CA1_sv_test, CA1_isi_test, CA1_ism_test])
```

Like a single test, a test suite is capable of judging one or more models. The result is a score matrix much like the one diagramed in Fig. 1.

```python
CA1_matrix = CA1_suite.judge([CA1_linear_model_heuristic, CA1_linear_model_fit])
```

A simple summary of the scores in a score matrix can be printed to the console or visualized by other tools, such as the web application *SciDash* described in Sec. 4.

3 INTEGRATING WITH NEUROINFORMATICS TOOLS AND INFRASTRUCTURE

In the examples above, tests were parameterized by data explicitly, and models implemented capabilities directly. In many cases, however, both models and data can be accessed from specialized repositories in standardized formats. In this section, we will show how to leverage this existing infrastructure to make parameterizing tests with published data and generating *SciUnit* models from models implemented in some other manner (even in a language other than Python) nearly automatic.

3.1 *NEURONUNIT*: NEUROPHYSIOLOGY-SPECIFIC TESTING TOOLS

In particular, we will focus on neurophysiology in this paper as our initial case study, using machine-readable models written in *NeuroML* available from OpenSourceBrain (*OSB*), and machine-readable data from resources like *NeuroElectro*. We will provide functions that link these tools to *SciUnit*, as a collection of relevant model and test families and associated capabilities, in a package called *NeuronUnit*, collaboratively developed on Github (*http://github.com/scidash/neuronunit*) . Scientists can use these tools to create test suites tailored to specific systems of interest (for example, scientists interested specifically in models of cells in the hippocampus would use *NeuronUnit* to create a suite, *HippocampusSuite*, parameterized by relevant data from *NeuroElectro*).

Must change name of repository on github to neuronunit (af- ter J

```python
class CA1PyramidalCellModel(NeuroConstructModel):
    """CA1 Pyramidal Cell model from Open Source Brain."""
    def __init__(self,**kwargs):
        project_path = neuroconstruct.get_path("hippocampus","CA1_pyramidal_neuron","CA1PyramidalCell"
            ,"neuroConstruct")
        models.NeuroConstructModel.__init__(self,project_path,**kwargs)
```

Figure 6. A model class corresponding to a CA1 Pyramidal Cell model from Open Source Brain

We anticipate that other communities within neuroscience (and indeed, within science more broadly) will develop similar repositories that synergistically link *SciUnit* to relevant ongoing standardization efforts and provide a common, collaboratively-developed vocabulary of test families, model families and capabilities relevant to their domain.

3.2 MODELS FROM NEUROML

NeuroML is a standardized model description language for neuroscience (**Gleeson et al.**, 2010). It permits many neurophysiological/neuroanatomical models to be described in a simulator-independent fashion, and executed across many popular simulators due to inter-conversion capabilities of the NeuroML API. Because NeuroML is an XML specification, model descriptions can be validated for correctness and queried for model properties and components, exposing potential capabilities. It is ideal for model sharing, curation, and for answering both *what* and *how* programatically.

NeuroConstruct (**Gleeson et al.** (2007), *http://www.neuroconstruct.org*) is a simulation manager that takes NeuroML models and hands off simulation to supported simulators. *NeuronUnit* offers a `sciunit.Model` subclass called `NeuroConstructModel`, instantiated with the path to a NeuroML model. Because NeuroML can describe such a wide range of models, `NeuroConstructModel` makes few assumptions about them: that each one is `TimeIntegrable`, and `HasMembranePotential`. It is subclassed to test *specific* NeuroML models (Fig. 6).

OSB curates many models described in NeuroML. OSB-curated projects are converted from their native format into NeuroML, and run on major neural simulators such as Neuron (**Carnevale and Hines** (2006), *http://www.neuron.yale.edu/neuron*), GENESIS (**Bower and Beeman** (2007), *http://genesis-sim.org*), NEST (**Gewaltig and Diesmann** (2007), *http://www.nest-initiative.org*), and MOOSE (**Ray et al.** (2008), *http://moose.ncbs.res.in*). Concordance between model output (beginning with the NeuroML description) and reference output (from native simulator source files) is reported for each model. Thus, OSB is an excellent source of models that, in addition to being open source, are sufficiently described to enable validation. The hippocampal CA1 pyramidal cell is commonly modeled, and we implement one such model hosted on OSB (*http://opensourcebrain.org/projects/ca1pyramidalcell*) by simply declaring a `CA1PyramidalCellModel` class, inheriting from `NeuroConstructModel`. This basic implementation simply "wraps" the components of the existing model, with simulator interaction taken care of by `NeuroConstructModel` methods; thus, only the code shown in Fig. 6 is required. All OSB models, and indeed any NeuroML model, can be tested similarly.

Spanning a range of scales and original development environments, all OSB models are formally described using NeuroML, as are all model components and sub-components, such as cells, ion channels, calcium stores, etc. These models are regularly executed on OSB servers to ensure that their output remains consistent as they are updated. Therefore, OSB can confirm that they *do* work, while linked journal articles, on-site wiki, and code inspection can establish *how* they work. However, there is no mechanism for establishing *how well* they work, i.e. how well the models accord with data. *NeuronUnit* fills this gap by helping OSB (and the larger biology community) assess models using data-driven tests. *SciUnit* can be applied similarly to other neuroscience (or biology) sub-disciplines using *NeuronUnit* analogues written by the corresponding communities.

3.3 CAPABILITIES FROM NEUROTOOLS

NeuroTools (*http://neuralensemble.org/NeuroTools*) is a Python library supporting tasks associated with analysis of neural data (or model output), such as membrane potential time series, spike trains, etc. It is an open source and actively developed project, containing reliable algorithms on which to base neurophysiology tests.

We use NeuroTools to implement *SciUnit* capabilities in *NeuronUnit*. For example, an `AnalogSignal` object (e.g. a membrane potential time series) from NeuroTools has a threshold detection method that returns a NeuroTools `SpikeTrain` object. A *NeuronUnit* `HasSpikeTrain` Capability requires that the method `getSpikeTrain` be implemented. `NeuroConstructModel` does so by placing the object method `AnalogSignal.threshold_detection` inside `getSpikeTrain`. Many such NeuroTools objects are similarly exchanged between `NeuroConstructModel` methods. This simplifies test writing, since basic model output properties are obtained trivially using NeuroTools object methods, and these NeuroTools objects are easily extracted from model output using candidate models subclassing `NeuroConstructModel`.

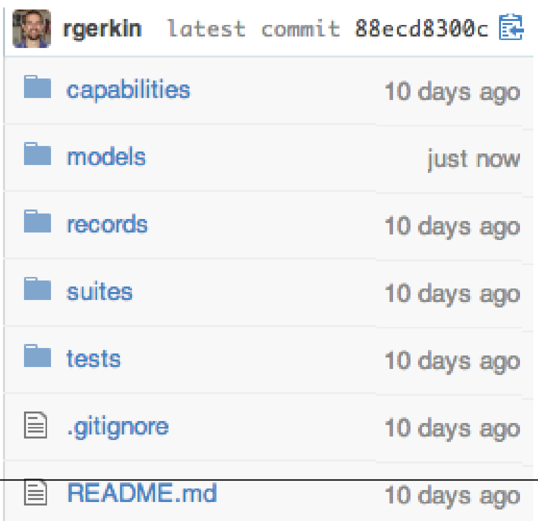
3.4 REFERENCE DATA FOR TESTS FROM NEUROELECTRO

Answering *how well* requires validation testing against data. The NeuroElectro project (*http://neuroelectro.org*) is an effort to curate all published single cell neurophysiology data (**Tripathy et al.**, 2012). Currently, up to 27 electrophysiological properties are reported for 93 cell types, spanning > 2000 single pieces of published data extracted from article tables. We have made it easy to construct *NeuronUnit* tests using the NeuroElectro API to get reference data. Tests can be based upon data from single journal articles, or from ensembles of articles with a common theme (e.g. about a particular neuron type). The former is illustrated in Figure 9. Associated statistics of that data (e.g mean, standard error, and sample size) are attached and enable judgement of model output according to a chosen scoring mechanism. While NeuroElectro alone cannot judge all model aspects, it can serve to validate basic features of many neurophysiology models, such as resting membrane potential, action potential width, after-hyperpolarization amplitude, etc. As NeuroElectro is the only publicly curated source of such data, it represents a key component for *NeuronUnit* test constuction.

4 *SCIDASH*: A COMMUNITY WEB APPLICATION

A collection of tests, models, and test records can summarize the current state of modeling in a research area. Community-oriented cyberinfrastructure to support the creation and summarization of such collections is essential. We used a service called *SciDash* (*http://www.scidash.org*) for coordinated development of software repositories.

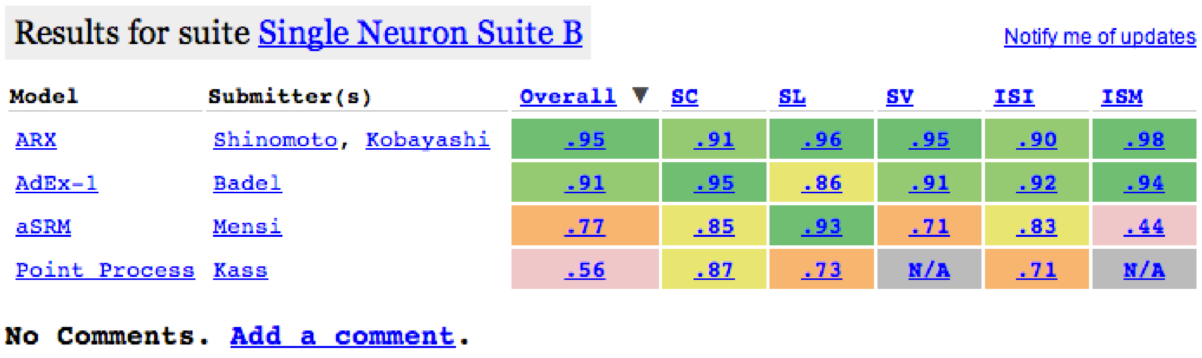
SciDash exists in three layers. The **first layer** is completely under the control of individual developer communities, and consists of git repositories (**Ram**, 2013) stored on GitHub *http://github.com*. *SciDash* identifies all similarly structured repositories (by tracking the lineage of a vanilla *SciDash* repository above using the GitHub API). This list of repositories is indexed by *SciDash* to form the **second layer**, an overview of the state of all *SciDash* repositories on GitHub. This index is searchable and filterable on the *SciDash* website to identify repositories of interest to a particular research community. The **third layer** is a "dashboard" view of the state of a field, visible



rgerkin latest commit 88ecd8300c

capabilities	10 days ago
models	just now
records	10 days ago
suites	10 days ago
tests	10 days ago
.gitignore	10 days ago
README.md	10 days ago

Figure 7. A *SciDash* repository on GitHub



Results for suite Single Neuron Suite B

Notify me of updates

Model	Submitter(s)	Overall ▼	SC	SL	SV	ISI	ISM
ARX	Shinomoto, Kobayashi	.95	.91	.96	.95	.90	.98
AdEx-1	Badel	.91	.95	.86	.91	.92	.94
aSRM	Mensi	.77	.85	.93	.71	.83	.44
Point Process	Kass	.56	.87	.73	N/A	.71	N/A

No Comments. Add a comment.

Figure 8. A *SciDash* record matrix

```python
# Interface with neuroelectro.org to search for spike widths of CA1 Pyramidal cells.
reference_data = NeuroElectroSummary(neuron={'id':85}, ephysprop={'id':23})
reference_data.get_values()  # Get summary data for the above.
model = CA1PyramidalCellModel(population_name="CG_CML_0") # Initialize the model with some parameters.
test = SpikeWidthTestDynamic( # Initialize the test.
    reference_data = {'mean':reference_data.mean, 'std':reference_data.std}, # Summary statistics from
        the reference data
    model_args = {'current':40.0}, # Somatic current injection in pA.
    comparator = ZComparator), # A comparison class that implements a Z-score.
result = sciunit.judge(test,model) # (1) Check capabilities, (2) take the test, (3) generate a score
    and validate it, (4) bind the score to model/test combination.
result.summarize() # Summarize the result.
```

Figure 9. Working example of testing in *NeuronUnit*

through a matrix of records for model/test combinations in a GitHub repository (Fig. 7). Test records are visualized as a "record matrix" composed of large numbers of model/test combinations (Fig. 8). Each row in this matrix contains results for all tests taken by one model and serve as clear evidence of that model's scope and validity. Hyperlinks in the record matrix point to models, tests, and records available on GitHub. Each record links to test records (stored in the repository), displaying line-by-line the executed code and intermediate output statements, as well as the underlying models and tests.

5 DISCUSSION

5.1 A COMPLETE PIPELINE

Although the tools described in Sec. 3.1 do not exhaust the possible sources of models, capabilities, and test data, they provide an immediate point of entry into the neurophysiology community and a powerful demonstration of our proposal. In the *NeuronUnit* repository (*http://github.com/scidash/neuronunit*) is a runnable script (*examples.py*) demonstrating a complete testing pipeline. It (1) selects an OSB model; (2) simulates it using NeuroConstruct; (3) tests the widths of the resulting action potentials, extracted and computed using NeuroTools, against NeuroElectro data downloaded on-the-fly, using a *NeuronUnit* test class called `SpikeWidthTestDynamic`; and (4) computes and prints a test score (Figure 9).

5.2 CREATING NEW MODELS, CAPABILITIES, AND TESTS

NeuronUnit provides base classes to enable rapid generation of models, capabilities, and tests for neurophysiology data. However these objects can also be created from scratch, requiring only adherence to the *SciUnit* interface. For example, a Model could implement an `integrate` capability method by wrapping execution of a MATLAB script and a `get_spikes` capability method by parsing a .csv file on disk; a Test could be initialized using empirical spike rates collected in the lab. While this does not meet our idealized vision of development and testing, in practice this may be a common scenario. Adoption of *SciUnit* into traditional, custom workflows would be reflected in the Methods sections of articles.

5.2.1 Collaboration Community-moderated comments on GitHub will allow test-makers and test-takers to discuss issues associated with test suites. On GitHub, these takes the form of "issues," commit messages, and comments surrounding merge requests from forked repositories. Thus, disagreements about the appropriateness of a test can be openly aired and in many cases resolved. The *SciDash* website itself can also support public comment to extend the features of GitHub. More importantly, we will enable sorting and filtering of *SciDash* results by repository statistics, e.g. the volume of activity and the number of followers, via the GitHub API. This will allow the most important test suites, as judged by each community, to be featured prominently on *SciDash*. Simple tagging should enable filtering by subject matter. We also will support open authentication via existing web applications (Google, Twitter, etc.), lowering the barrier to participation.

5.3 PARTICIPATION FROM MODELING COMMUNITIES

Modelers may not want to expose model capabilities, a requirement of test-taking. We anticipate four solutions: **First**, interfacing a model to *SciUnit* requires only implementing selected model capabilities. Often this means identifying native model procedures that satisfy a capability, and wrapping their functionality. This can require as little as one line of code. Importantly, the modeler is not required to expose or rewrite any model flow control. **Second**, we support multiple environments automatically by using NeuroML (**Gleeson et al.**, 2010), and other simulator-independent model descriptions are possible for other domains. Automated generation of NeuroML from native model source code is in development (Gleeson, personal communication); for the popular NEURON simulator (**Carnevale and Hines** (2006)), this functionality is already mature and in use. This minimizes modeler effort for a large (at least 1000, *http://www.neuron.yale.edu/neuron/node/69*) and growing number of neuronal models. **Third**, modelers have an incentive to demonstrate publicly their models' validity. Participation in public modeling competitions demonstrates this incentive. **Fourth**, modelers have an incentive to use *SciUnit* during development (see TDD, above) to ensure that ongoing development preserves correspondence between model and data. A popular test suite can represent a "gold standard" by which progress during development is judged.

5.4 PARTICIPATION FROM EXPERIMENTAL COMMUNITIES

Experimentalists may not want to write tests derived from their data. We anticipate four solutions: **First**, tests require no special data formatting; only a list of required capabilities (for selecting eligible models), optional metadata (as run-time arguments), and a statistical data summary (for scoring tests) are required. A unit test is focused and does not require arbitrary computations on data. For example, suppose intracellular current injection evokes 100 action potentials, the width of which is of interest. Writing the test consists of selecting `ReceivesCurrent` and `ProducesActionPotentialShape` capabilities (one line of code each), computing the mean and variance of action potential widths (one line of code), specifying current injection parameters, e.g. amplitude and duration (two lines of code), and selecting a scoring mechanism from `sciunit.scores`, e.g. (colloquially) "Must be <1 standard deviation of the mean" (one line of code). This example can be found in `NeuronUnit.tests.SpikeWidthTest`; heavy-lifting is done by the interface. **Second**, data-sharing is becoming accepted, and test-writing can be distributed across scientists, including non-experimentalists with other aims such as analysis or modeling.

Third, many tests can be automatically generated using the NeuroElectro API, and the continued emergence of such data-aggregation initiatives will expand these possibilities. **Fourth**, an incentive to write tests for one's data exists: the ability to identify models that give the data clear context and impact.

5.5 DIVERSITY OF LEVELS AND KINDS OF MODELS AND DATA

The diversity of topics in biology is vast. **First**, we address this by providing an interface allowing modelers to express specific capabilities. This capability set determines the range of eligible tests. Scale hierarchies are embedded in capability inheritance. = For example, `HasActionPotentials` inherits from `HasNeurons`, and `HodgkinHuxley` inherits from `VoltageGated`. Thus, incompatibility of a test-requiring-action-potentials for a model-lacking-neurons is known without explicit tagging. **Second**, NeuroML naturally addresses diversity of scales because it is organized hierarchically, in "levels." Models can be sub- or supersets of other models; similarly for SBML (**Hucka et al.** (2003), *http://sbml.org*) a general systems biology markup language. **Third**, cross-level testing can use "Representional Similarity Analysis" (RSA) (**Kriegeskorte et al.**, 2008), requiring only that a model respond to defined inputs (e.g. stimuli). A "similarity matrix" for input responses defines a unique model signature, and can serve as intermediate test output. Goodness-of-fit between similarity matrices for model and experiment determines test scores; these matrices are independent of model scale because their size depends only on test inputs, not system detail.

5.6 ARBITRARY SCORING CRITERIA FOR TESTS

A test first assesses goodness-of-fit, and applies a normalization (e.g. pass/fail, 0.0-1.0) to generate a score. Arbitrary choices at both stages may benefit some models over others. **First**, however, rank-ordering is constant across many goodness-of-fit metrics, meaning that choice of metric will rarely cause an otherwise passing model to fail and vice versa. For example, given a data mean and variance, ordering model output by Z-score or p-value will yield the same relative ranking of models. Indeed, rank ordering of models may prove more valuable than test scores themselves. **Second**, suite repositories are open (e.g. Fig. 7), so tests can be cloned and new statistical choices implemented. Statistics as applied to neuroscience have been notoriously "fast and loose"; identification and correction of flawed methodology is becoming increasingly common (**Button et al.** (2013); **Kriegeskorte et al.** (2009); **Galbraith et al.** (2010), *http://prefrontal.org/files/posters/Bennett-Salmon-2009.jpg*), and is accelerated by an open framework. The validation criteria is made explicit in the specification of a test, so modelers need not guess which criteria are being used to validate or invalidate their model. Validation criteria are subject to debate (indeed, the QSNMC criteria changed between 2007 and 2008 due to such debates), and neuroscientists who wish to promote different criteria need only derive alternative tests. A community consensus will slowly emerge in the form a commonly-accepted test suite (see Sec. 4). This community can judge which test version is most appropriate, i.e. what a model *should* do – this process documented via standard moderation techiniques used on GitHub – and the *SciUnit* framework determines whether the model *does* it. This process can be made even more visible by storing the output of test runs using Sage or IPython worksheets, or through optional database backends such as in the Sumatra lab notebook for simulation and analysis (**A.P.** (2012), *http://neuralensemble.org/sumatra*).

5.7 RELIABILITY OF DATA UNDERLYING TESTS

Unreliable data can undermine model validation. **First**, the community must evaluate experimental design and methods, discounting data produced using questionable techniques. GitHub supports community moderation, permitting users to comment on tests, indicating their concerns. Suite repository popularity, by which *SciDash* results can be filtered, can reflect consensus. Experimental metadata also constrains a test's relevance, so test writers should select data with metadata appropriate to the system being modeled, and attach the metadata to resulting test scores. Metadata can also be expressed as Capabilities, e.g. `At37Degrees` or `Calcium3mM`; and tests can require that models express them. Such capabilities require no implementation, so the model definition must only inherit them. **Second**, models cannot perfectly reproduce data that is itself a random draw from a "true" distribution. Uncertainty in data must be made explicit, by asking how well a data set validates its own experimental replications (**Kriegeskorte et al.**, 2008). The degree of such "self-validation" represents the upper limit of what a model can be expected to achieve, and should represent a "perfect" score.

5.8 OCCAM'S RAZOR

All things being equal, simpler models are better. Model complexity has many definitions, so *SciDash* will report several complexity metrics (**McCabe**, 1976), including: 1) model length; 2) memory use; 3) CPU load; 4) # of capabilities. *SciDash* will report the model validity vs complexity tradeoff in tabular form (e.g. Table 8), and in a scatter plot, with the "best" models being in the high validity / low complexity corner of the plot. The set of models which *dominate* all others, i.e. that have the highest validity for a given complexity, can be represented as a "frontier" in such a scatter plot, a visualization familiar from the symbolic regression package Eureqa (**Schmidt and Lipson**, 2009).

5.9 EXPANSION INTO OTHER AREAS OF BIOLOGY

After covering neurophysiology, we would like *SciUnit* to be applied across neuroscience and in other biological sciences. The framework is discipline-agnostic, so community participation and model description are the only obstacles. Community participation begins with enumerating the capabilities relevant to a sub-discipline, and then writing tests. Model description can expand within NeuroML (which already covers multiple levels within neuroscience) and tools for capability implementation can incorporate libraries for neuroimaging (NiBabel, *http://nipy.org/nibabel*), neuroanatomy (NeuroHDF, *http://neurohdf.readthedocs.org*) and other sub-disciplines. SBML will enable expansion beyond neuroscience, facilitated by parallel efforts among NeuroML developers to interface with it (Crook, unpublished). One intriguing possiblity is applying *SciUnit* to the OpenWorm project (*http://www.openworm.org*), which through open, collaborative development seeks to model the entire organism C. elegans.

DISCLOSURE/CONFLICT-OF-INTEREST STATEMENT

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

ACKNOWLEDGEMENTS

We thank Sharon Crook, Jonathan Aldrich, Shreejoy Tripathy, and Padraig Gleeson for their support of this project. The work was supported in part by grant R01MH081905 from the National Institute of Mental Health. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

REFERENCES

A.P., D. (2012), Automated capture of experiment context for easier reproducibility in computational research environment, *Computing in Science and Engineering*, 14, 48–56

Beck, K. (2003), Test Driven Development: By Example (Addison Wesley)

Bower, J. M. and Beeman, D. (2007), Constructing realistic neural simulations with GENESIS, *Methods Mol. Biol.*, 401, 103–125

Button, K. S., Ioannidis, J. P. A., Mokrysz, C., Nosek, B. A., Flint, J., Robinson, E. S. J., et al. (2013), Power failure: why small sample size undermines the reliability of neuroscience, *Nature reviews. Neuroscience*, 14, 5, 365–376, doi:10.1038/nrn3475, PMID: 23571845

Carnevale, N. and Hines, M. (2006), The NEURON Book (Cambridge University Press, Cambridge, UK)

Fisher, R. A. (1925), Statistical Methods for Research Workers (Oliver and Boyd, Edinburgh)

Galbraith, S., Daniel, J. A., and Vissel, B. (2010), A study of clustered data and approaches to its analysis, *The Journal of Neuroscience*, 30, 32, 10601–10608, doi:10.1523/JNEUROSCI.0362-10.2010, PMID: 20702692

Gewaltig, M.-O. and Diesmann, M. (2007), Nest (neural simulation tool), *Scholarpedia*, 2, 4, 1430

Gleeson, P., Crook, S., Cannon, R. C., Hines, M. L., Billings, G. O., Farinella, M., et al. (2010), NeuroML: a language for describing data driven models of neurons and networks with a high degree of biological detail, *PLoS computational biology*, 6, 6, e1000815, doi:10.1371/journal.pcbi.1000815, PMID: 20585541

Gleeson, P., Piasini, E., Crook, S., Cannon, R., Steuber, V., Jaeger, D., et al. (2012), The open source brain initiative: enabling collaborative modelling in computational neuroscience, *BMC Neuroscience*, 13, Suppl 1, O7, doi:10.1186/1471-2202-13-S1-O7, PMID: null PMCID: PMC3403499

Gleeson, P., Steuber, V., and Silver, R. A. (2007), neuroConstruct: a tool for modeling networks of neurons in 3D space, *Neuron*, 54, 2, 219–235, doi:10.1016/j.neuron.2007.03.025, PMID: 17442244

Hopfield, J. J. and Brody, C. D. (2000), What is a moment? ”Cortical” sensory integration over a brief interval, *Proceedings of the National Academy of Sciences of the United States of America*, 97, 25, 13919–13924, doi:10.1073/pnas.250483697, PMID: 11095747

Hucka, M., Finney, A., Sauro, H. M., Bolouri, H., Doyle, J. C., Kitano, H., et al. (2003), The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models, *Bioinformatics (Oxford, England)*, 19, 4, 524–531, PMID: 12611808

Jinha, A. E. (2010), Article 50 million: an estimate of the number of scholarly articles in existence, *Learned Publishing*, 23, 3, 258–263, doi:10.1087/20100308

Jolivet, R., Schrmann, F., Berger, T., Naud, R., Gerstner, W., and Roth, A. (2008), The quantitative single-neuron modeling competition, *Biological Cybernetics*, 99, 4, 417–426, doi:10.1007/s00422-008-0261-x

Kriegeskorte, N., Mur, M., and Bandettini, P. (2008), Representational similarity analysis - connecting the branches of systems neuroscience, *Frontiers in systems neuroscience*, 2, 4, doi:10.3389/neuro.06.004.2008, PMID: 19104670

Kriegeskorte, N., Simmons, W. K., Bellgowan, P. S. F., and Baker, C. I. (2009), Circular analysis in systems neuroscience: the dangers of double dipping, *Nature neuroscience*, 12, 5, 535–540, doi:10.1038/nn.2303, PMID: 19396166

McCabe, T. (1976), A complexity measure, *IEEE Transactions on Software Engineering*, 2, 4, 308–320

Ram, K. (2013), Git can facilitate greater reproducibility and increased transparency in science, *Source code for biology and medicine*, 8, 1, 7, doi:10.1186/1751-0473-8-7, PMID: 23448176

Ray, S., Deshpande, R., Dudani, N., and Bhalla, U. S. (2008), A general biological simulator: the multiscale object oriented simulation environment, moose, *BMC Neuroscience*, 9, Suppl. 1, 93

Schmidt, M. and Lipson, H. (2009), Distilling free-form natural laws from experimental data, *Science (New York, N.Y.)*, 324, 5923, 81–85, doi:10.1126/science.1165893, PMID: 19342586

Tripathy, S., Saviskaya, J., Gerkin, R., and Urban, N. (2012), NeuroElectro: a database describing the electrophysiology properties of different neuron types, *Neuroinformatics Meeting*