Frontiers in NeuroInformatics

Methods
March 27, 2014

Collaboratively Testing the Validity of Neuroscientific 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*
The authors contributed equally.

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

Python in Neuroscience II

ABSTRACT

Rigorously validating a quantitative scientific model requires comparing its predictions against an unbiased selection of experimental observations according to sound statistical criteria. Developing new models thus requires a comprehensive and contemporary understanding of competing models, relevant data and statistical best practices. Today, developing such an understanding requires an encyclopedic knowledge of the literature. Unfortunately, in rapidly-growing fields like neuroscience, this is becoming increasingly untenable, even for the most conscientious scientists. For new scientists, this can be a significant barrier to entry.

Software engineers seeking to verify, validate and contribute to a complex software project rely not only on volumes of human documentation, but on suites of simple executable tests, called “unit tests”. Drawing inspiration from this practice, we have developed *SciUnit*, an easy-to-use framework for developing “model validation tests” – executable functions, here written in Python. These tests generate and statistically validate predictions from a specified class of scientific models against one relevant empirical observation to produce a score indicating agreement between the model and the data. Suites of such validation tests, collaboratively developed by a scientific community in common repositories, can produce up-to-date statistical summaries of the state of the field. In this paper, we aim to detail this test-driven workflow and introduce it to the neuroscience community. As an initial example, we describe *NeuronUnit*, a library that builds upon *SciUnit* and integrates with several existing neuroinformatics resources to support validating single-neuron models using data gathered by neurophysiologists.

Keywords: Neuroinformatics Simulation Electrophysiology Software Modeling Validation

1 INTRODUCTION

1.1 THE PROBLEM: WHAT DO MODELS DO AND HOW WELL DO THEY DO IT?

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

Today, scientists contribute a new model to the research community by submitting a paper containing a description of the model's structure and scope along with text and figures that demonstrate its validity and argue for its novelty. The scientists tasked with reviewing the paper are responsible for evaluating these claims, discovering competing models and relevant data the paper did not adequately consider, and ensuring that goodness-of-fit was measured in a statistically sound manner, drawing on their knowledge of prior literature. Often, there are no means for verifying even the basic results (**Donoho et al.**, 2008). And once a modeling paper is published, it is frozen and cited as-is in perpetuity.

Because publications are the sole vehicle for knowledge about model scope and validity, scientists must rely on an encyclopedic knowledge of the literature to answer *summary questions* like the following:

- Which models are capable of predicting the quantities I am interested in?
- What are the best practices for evaluating the goodness-of-fit between these models and data?
- How well do these models perform, as judged by these metrics, given currently available data?
- What other quantities can and can't these models predict?
- What observations have not yet been adequately explained by any available model?

In some fields, like neuroscience, where the number of relevant publications being generated every year is growing rapidly (**Jinha**, 2010), these questions can be difficult for even conscientious senior scientists to answer comprehensively. For new scientists, this represents a serious barrier to entry.

1.2 THE SOLUTION: UNIT TESTING FOR MODELS

Professional software developers face similar issues (**?**). They must understand the scope of each component of a complex software project and verify that each component achieves desired input/output behavior. But software developers do not verify components by simply documenting a few interesting inputs and corresponding outputs and then presenting them to other developers for one-time review leading to an archived document. 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 verify its implementation on an ongoing basis 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 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, in the form of *modeling competitions*. During these competitions, competitors develop and parameterize models based on publicly-available training data and submit them to a central server. There, submitted models are provided hidden testing data which they must use to produce predictions. These predictions are validated using publicly available criteria to produce summaries of the relative merits of different models, just as a test suite summarizes the state of a software project. Such competitions continue to drive important advances and improve scientists' understanding of their fields. For example, the quantitative single neuron modeling competition (QSNMC) (**Jolivet et al.**, 2008) investigated 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 areas of biomedical research abound (e.g. *http://www.the-dream-project.org*).

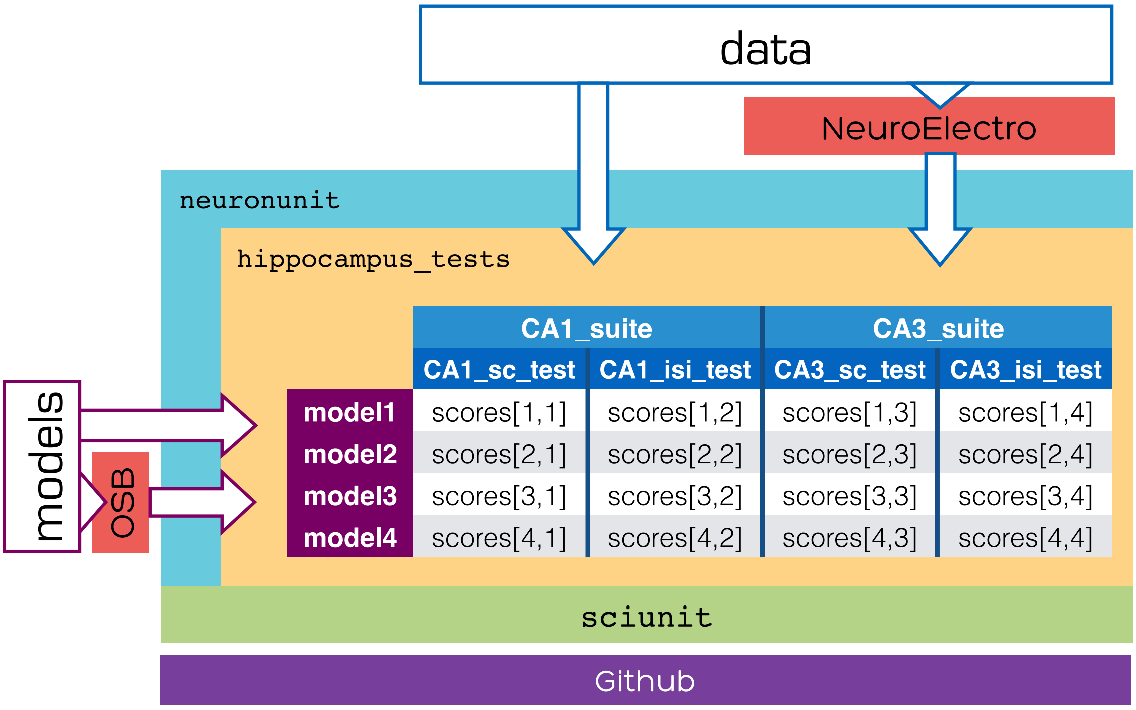

Figure 1. Paper overview. NeuronUnit is set of neurophysiology-specific testing tools built upon the domain-agnostic SciUnit framework. Scientists interested in testing neurophysiological models of particular systems, like the hippocampus, against relevant experimental data can construct test suites in a repository called *hippocampus_tests*. Models and data can be added directly or imported, via NeuronUnit, from model repositories like Open Source Brain (OSB) and data repositories like NeuroElectro. Testing tools and test repositories are developed collaboratively using Github.

1.3 THE IMPLEMENTATION: SCIUNIT AND NEURONUNIT

Each of these examples has leveraged *ad hoc* infrastructure to support model validation. While the specific criteria used to evaluate models can vary widely between modeling domains, the underlying methodology is common and could be implemented once. Recognizing this, and inspired by unit testing practices, we have developed a discipline-agnostic framework for developing *model validation test suites* called *SciUnit* (**?**) (available from *http://sciunit.scidash.org*). In this paper, we will begin by detailing *SciUnit*, focusing on examples from single-neuron physiology (Sec. 2).

SciUnit contains validation logic common across scientific disciplines. But a particular discipline, such as neurophysiology, might have more specialized logic associated with it that can be shared amongst its sub-disciplines (e.g. hippocampal neurophysiology). We anticipate a collaborative workflow where this common logic is developed in common repositories on a social coding service, here *GitHub*. For neurophysiology, we have developed such a repository, called *NeuronUnit*. This repository contains common testing logic as well as bridges to existing informatics infrastructure to make it easy to import existing data and models. In particular, we will show 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 a fully automated fashion using published data curated by the *NeuroElectro Project* (NeuroElectro, **Tripathy et al.** (2012), *http://neuroelectro.org*), leveraging facilities from the *NeuroTools* library (*http://neuralensemble.org/NeuroTools*) to extract features from model outputs (Sec. 4). Scientists in particular sub-disciplines, like hippocampal neurophysiologists, can use this common logic to select relevant models and tests, forming focused test suites in common repositories. The end result of this workflow is a table like the one shown in Figure 1, where submitted models can be compared based on the scores they achieve on different selected tests. These tables serve as a summary of the status of a modeling community, just as the results from a suite of unit tests serve as a summary of the status of a software project, and help scientists answer the questions mentioned earlier in this section more easily and accurately.

```python
class SpikeCountTest(sciunit.Test):
  """Tests spike counts produced in response to several current stimuli against observed means and standard deviations.

  goodness of fit metric: Computes p-values based on a chi-squared test statistic, and pools them using Fisher's method.
  parameters:
    inputs: list of numpy arrays containing input currents (pA)
    means, stds: lists of observed means and standard deviations, one per input
  """
  def __init__(self, inputs, means, stds):
    self.inputs, self.means, self.stds = inputs, means, stds

  required_capabilities = [SpikeCountFromCurrent]

  def _judge(self, model):
    inputs, means, stds = self.inputs, self.means, self.stds
    n = len(inputs)
    counts = numpy.empty((n,))
    for i in xrange(n):
      counts[i] = model.spike_count_from_current(inputs[i])
    chisquared = sum((counts-means)**2 / means) # An array of chi-squared values.
    p = scipy.stats.chi2.cdf(chisquared,n-1) # An array of p-values.
    pooled_p = sciunit.utils.fisherp(p_array) # A pooled p-value.
    return sciunit.PValue(pooled_p, related_data={
      "inputs": inputs, "counts": counts, "empirical_means": means, "empirical_stds": stds
      })
```

Figure 2. `[SpikeCountTest]` 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. 4).

2 VALIDATION TESTING WITH *SCIUNIT*

2.1 EXAMPLE: THE QUANTITATIVE SINGLE NEURON MODELING COMPETITION

We begin by building an 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 of a particular subtype, while the somatic membrane potential of each stimulated cell (in mV) is recorded and stored. A model claiming to capture this neuron 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

```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. `[SpikeCountFromCurrent]` 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. `[TrainSpikeCountFromCurrent]` Another capability specifying a training protocol (not used by the test in Figure 2).

`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 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 CA1 cells (not shown), we can instantiate a test as follows:

```
In [0]: 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. Test generation for a particular system of interest will then 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. 4). Particular tests, like `CA1_sc_test`, will be derived from these families in suite repositories (e.g. `hippocampus_tests`, as shown in Figure 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, 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 needs a model to implement so that it can provide inputs and generate relevant predictions, 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.

The remainder of the `_judge` method implements the goodness-of-fit metric described above, returning an instance of `sciunit.scores.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 empirical (experimental) 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.

```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. `[LinearModel]` 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).

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 provides parameter values, just as with test families:

```
In [1]: 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 alternative 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 capable of being automatically trained (like `LinearModel`) can simply be trained explicitly by calling the capability method just like any other Python method:

```
In [2]: CA1_linear_model_fit = LinearModel()
In [3]: 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:

```
In [4]: score = CA1_sc_test.judge(CA1_linear_model_heuristic)
```

`judge` is a built-in method of `sciunit.Test` that proceeds by first checking that the provided model implements all required capabilities. It then calls the test's author-implemented `_judge` method (note leading underscore) to produce a score. A score is an instance of `sciunit.Score` and must implement an ordering, to facilitate sorting of scores (e.g. in tabular form, described below). 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 test suite could be used for a simplified version of the QSNMC, as described in sec. 2.1:

```
In [5]: CA1_suite = sciunit.TestSuite([CA1_overall_test, CA1_sc_test, CA1_sl_test, CA1_sv_test,
            CA1_isi_test, CA1_imi_test])
```

Like a single test, a test suite is capable of judging one or more models. The models must possess the union of the capabilities of the tests in a suite to be fully capable of running the suite, though partial results can also be generated. When a model cannot take a test because of a missing capability, the `NotApplicable` score is used (e.g. point processes cannot predict subthreshold voltages). The result is a *score matrix*, like the one diagrammed in Fig. 1.

```
In [7]: scores.show_table()

Out[7]:
```

Model	Submitter(s)	Overall ▼	SC	SL	SV	ISI	IMI
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	.50

Figure 6. A score matrix for a SciUnit test suite, visualized as a hyperlinked table inside an IPython notebook stored in a test repository like *hippocampus_tests*.

```
In [6]: scores = CA1_suite.judge([ARX_model, AdEx1_model, aSRM_model, PP_model])
```

A simple summary of the scores in a score matrix can be printed to the console or visualized by other tools. For example, the score matrix can be visualized as a hyperlinked table inside an IPython notebook (**Pérez and Granger**, 2007) (Fig. 6). The code for generating such visualizations is available in the main SciUnit repository (*http://github.com/scidash/sciunit*), and for generating this particular visualization is available at *CyrusputaURLhere*.

3 A COLLABORATIVE WORKFLOW

The *NeuronUnit* package contains a collection of test families and associated capabilities relevant to neurophysiology, such as those described in the previous section. It is collaboratively developed on Github (*http://github.com/scidash/neuronunit*). Scientists can use this package to easily create test suites tailored to specific neuronal systems of interest. For example, scientists interested specifically in testing models of cells in the hippocampus would use *NeuronUnit* to create tests in a GitHub repository called *hippocampus_tests*, containing an IPython notebook and parameterized by empirical data from the hippocampus.

Validation criteria are subject to debate (indeed, the QSNMC criteria changed between 2007 and 2008 due to such debates), and identification and correction of flawed methodology is becoming increasingly common (**Button et al.** (2013); **Kriegeskorte et al.** (2009); **Galbraith et al.** (2010)). Statisticians and other scientists interested in improving how goodness-of-fit is measured can simply fork the *NeuronUnit* repository, modify the logic in the relevant test families and submit a *pull request* for evaluation by the community. Rather than attempting to persuade a large number of scientists that the methods used in canonical papers need to be changed, statisticians need only propagate a change to the testing logic used by the community. Once the testing logic has been changed, all models can be immediately evaluated against the new metrics. Scientists wishing to identify statistical best-practices for measuring goodness-of-fit need only refer to the tests in these repositories.

To submit a new model for testing, a scientist can similarly fork the *hippocampus_tests* repository, add a new model to the relevant suite and test the model against available tests locally. When satisfied with parameter choices, the scientist can submit a pull request to the common repository. Once a model is in the repository, it will be evaluated on an ongoing basis against the latest accepted collections of tests, parameterized by the latest data, using the latest statistical best practices, without the participation of the original modeler, assuming only that the capabilities required by the tests stay stable.

GitHub is an excellent platform for this kind of activity, and is becoming a widely-used tool for scientific collaboration, reproducibility, and transparency (**Ram**, 2013). We anticipate that other communities within neuroscience (and indeed, within science more broadly) will develop repositories similar to *NeuronUnit* that provide a common, collaboratively-developed vocabulary of test families and capabilities relevant to their domain, and bridge with relevant standardization efforts and infrastructure. Using these

```
1  reference_data = neuroelectro.NeuroElectroSummary( # Pooled experimental data from neuroelectro.org.
2    neuron = {'name':'Hippocampus CA1 pyramidal cell'}, # Neuron type according to NeuroLex ontology.
3    ephysprop = {'name':'Spike Width'}) # Electrophysiological property name in same.
4  reference_data.get_values()  # Get summary data for the above.
5  model = models.OSBModel( # Initialize the model.
6          'hippocampus', # Brain area.
7          'CA1_pyramidal_neuron', # Neuron type name in OSB.
8          'CA1PyramidalCell') # Model name (Migliore et al, 2005).
9  test = SpikeWidthTest( # Initialize the test.
10     reference_data = {'mean':reference_data.mean, 'std':reference_data.std}, # Summary statistics.
11     model_args = {'current':40.0*pA}, # Somatic current injection in pA.
12     comparator = ZComparator), # the scoring metric can be part of the test parameters
13 score = test.judge(model)
14 print score.summary
```

Figure 7. Working example of testing in *NeuronUnit*. Imports excluded for clarity.

common repositories, smaller groups of scientists interested in particular systems of interest will develop tests and models in test repositories like *hippocampus_tests*. Each research community can develop suitable quality standards and processes for merging pull requests.

The most visible result of these efforts will be tables, like the one shown in Figure 6, that allow scientists to represent, formally, the state of model development in their field, as judged against corresponding experimental data. We are developing a lightweight portal, *SciDash*, that directs scientists to the canonical test repositories in their field in order to prevent fragmentation and encourage community participation (**?**). *SciDash* will also facilitate viewing test results without needing to clone test repositories and run tests locally for exploratory purposes, based on the *iPython notebook* technology mentioned in Sec. 2.5. We leave details of *SciDash* as future work, focusing this paper on the core testing methodology (Sec. 2) and on the details of the bridge technologies, below.

4 INTEGRATING WITH NEUROINFORMATICS TOOLS AND INFRASTRUCTURE

In the examples in the previous sections, tests were provided data explicitly, and models implemented capabilities directly. In many fields, however, both models and data are available from existing community resources in standardized formats. In this section, we will show how to use this existing infrastructure to make 1) parameterizing tests with published data and 2) generating *SciUnit* models from models implemented in some other manner (even in a language other than Python) nearly automatic. In particular, we will continue to focus on neurophysiology as our initial case study, using machine-readable models described in *NeuroML* available from OpenSourceBrain (*OSB*), and machine-readable data from *NeuroElectro*.

4.1 REFERENCE DATA FOR TESTS FROM NEUROELECTRO

The NeuroElectro project (*http://neuroelectro.org*) is an effort to make all published data on single cell neurophysiology available in a machine-readable format (**Tripathy et al.**, 2012). Currently, up to 27 electrophysiological properties are available for 93 cell types, gathered from over 2000 single pieces of published data extracted from tables in published papers. NeuroElectro uses the NeuroLex.org ontology **Larson and Martone** (2013) to identify specific neuron types and specific electrophysiological properties, and indexes corresponding data values from the literature. We have made it easy to parameterize the test families in *NeuronUnit* using data extracted from the NeuroElectro API. 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 latter is illustrated in Figure 7, on lines 1-4. Once data has been retrieved from NeuroElectro, associated statistics (e.g. mean, standard error and sample size) can be extracted to parameterize tests (e.g. a test of spike widths on line 10). Data from NeuroElectro can be used to validate a variety of

```python
class CA1PyramidalCellModel(NeuroConstructModel):
    """CA1 Pyramidal Cell model in the local file system."""
    def __init__(self):
        project_path = neuroconstruct.get_path("hippocampus","CA1_pyramidal_neuron","CA1PyramidalCell"
            ,"neuroConstruct")
        NeuroConstructModel.__init__(self,project_path)
```

Figure 8. A model class corresponding to a CA1 Pyramidal Cell model (Migliore et al, 2005) downloaded locally from Open Source Brain

basic electrophysiological features of neuron models, including spike width, resting membrane potential, after-hyperpolarization amplitude, etc. As NeuroElectro is the only public, curated source of such data, it represents a key resource facilitating test construction. However, in selecting data from NeuroElectro (either averaged data across all reports, or data from selected reports) for test construction, one is implicitly assuming that it represents the kind data that ought to be binding in model development. For some models, data obtained in this way is ought not to guide model development, and other data sources will need to be identified and used for test construction.

4.2 MODELS FROM NEUROML AND OPENSOURCEBRAIN

NeuroML is a standardized model description language for neuroscience (**Gleeson et al.**, 2010). It allows many neurophysiological and neuroanatomical models to be described in a simulator-independent fashion. Some simulators, notably the popular NEURON package (**Carnevale and Hines** (2006), *http://www.neuron.yale.edu/neuron*), can seamlessly export model specifications as NeuroML. Because NeuroML is an XML specification, model descriptions can be verified for correctness and queried for model properties and components. We are working on leveraging the latter facilities to expose model capabilities automatically.

NeuroConstruct (**Gleeson et al.** (2007), *http://www.neuroconstruct.org*) is a simulation manager that takes NeuroML models and hands them off to a supported simulator for execution. A number of popular simulators support execution of NeuroML models, including 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*). *NeuronUnit* provides a `sciunit.Model` subclass called `NeuroConstructModel` that takes a path to a NeuroML model as a parameter. NeuroML can describe a wide range of models, but our `NeuroConstructModel` class currently supports only time-integrable neuron models with membrane potentials (exposing capabilities `TimeIntegrable` and `HasMembranePotential` at minimum; see Sec. 4.3). It can be instantiated directly or subclassed (useful when additional capabilities need to be manually exposed, for example) to generate a model suitable for testing (Fig. 8). The underlying simulator can be configured (defaulting to NEURON; not shown). In this example, we simply create a model based on a NeuroML model of CA1 cells **Migliore et al.** (2005) downloaded from OpenSourceBrain to the local file system upon instantiation (*http://opensourcebrain.org/projects/ca1pyramidalcell*).

OSB curates many models described in NeuroML. OSB-curated projects have been converted from their native format into NeuroML. To ensure that the NeuroML model is faithful to the original simulation, OSB verifies concordance between model output (beginning with the NeuroML description) and reference output (from native simulator source files or published figures) for each model. Thus, OSB is an excellent source of models that, in addition to being open source, are known to be accurately described. This also highlights the distinction between *verification* activities, performed by OSB to ensure that a model description operates correctly, and *validation* activites, performed using SciUnit to check the model's predictions against data. Both are important activities.

To directly test a model in OSB, we can skip the step of adding it to the test repository and simply download it from OSB directly by using the `OSBModel` class, which is a subclass of `NeuroConstructModel`, shown on lines 5-8 of Figure 7.

4.3 DATA FORMATS AND 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 a variety of *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. The *NeuronUnit* capability `HasSpikeTrain` requires that a method named `get_spike_train` be implemented, returning a `SpikeTrain` object. `NeuroConstructModel` implements this by default by calling the NeuroTools method `AnalogSignal.threshold_detection`, retrieving the membrane potential by calling the `get_membrane_potential` method implemented by the `HasMembranePotential` capability. This capability is implemented by the `NeuroConstructModel` class which supports all current OSB models and implicitly supports all NeuroML models with somatic membrane potentials. NeuroTools is used in this manner to implement a wide variety of capabilities for `NeuroConstructModel`s, without requiring that the modeler do so explicitly (though they can override these if they wish). This also helps scientists by directing them to a set of common utility functions and data object types that they can use, knowing that they are accepted by the community around NeuronUnit.

5 DISCUSSION

5.1 A COMPLETE PIPELINE

Although the tools described in Sec. 4 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 *hippocampus_tests* repository (*http://github.com/scidash/hippocampus_tests*) we are currently developing a complete test suite similar to the *ad hoc* suite used by the QSNMC, demonstrated by an IPython notebook (*ca1_suite.ipynb*). These tests will be parameterized by data drawn both from NeuroElectro and entered manually, as well as models drawn both from OSB and implemented manually. New tests, data and models can be added by contributing to these community resources and/or submitting pull requests; we anticipate handing control over merging these requests to members of the hippocampal modeling community once our interface stabilizes and this paper is published. This repository represents a template that other groups of systems neuroscientists can use for their system of interest, and a starting point for other research areas in neuroscience, and science more broadly.

5.2 MANUALLY 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 as exemplified in sec. 2. 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 resulting .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 collaborative development and testing, in practice this may be a common scenario.

5.3 PARTICIPATION FROM MODELING COMMUNITIES

Modelers may not want to formally expose their models to tests via capabilities, a requirement for test-taking. We anticipate four solutions: **First**, we will emphasize to the community that wrapping existing model code that satisfy a capability is quite simple, requiring as little as one line of code.

Importantly, the modeler is not required to expose or rewrite any internal model flow control in almost all cases. **Second**, we support multiple simulation 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 implemented 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 that this incentive can draw several participants. We plan to assist with such competitions by advocating for use of SciUnit as the underlying technical substrate in the future. Currently, we are in discussion with field specialists to organize competitions for spiking neuron models (a sequel to the QSNMC), spike-sorting algorithms, calcium imaging spike time reconstruction, brain-computer interfaces, neural network function prediction, and seizure prediction. **Fourth**, modelers have an incentive to use *SciUnit* during development (see TDD, Sec. 1) 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, even within a single project.

5.4 PARTICIPATION FROM EXPERIMENTAL COMMUNITIES

Experimentalists may not want to write tests derived from their data. We anticipate four solutions: **First**, we will emphasize that tests do not require releasing entire datasets; a test consists only of a list of required capabilities (for selecting eligible models), and sufficient statistics suitable for executing the scoring logic. This can simply be means and standard deviations in many cases, such as the examples in this paper. Most scoring logic will consist of a few calls to capability methods followed by a standard statistical test, which can be implemented once in *SciUnit* or a discipline-specific repository like *NeuronUnit*. **Second**, data-sharing is becoming accepted. If data is available from a community repository, as many granting agencies are increasingly requiring, then tests can often be generated automatically using bridges developed collaboratively by the community. **Third**, an incentive to write tests for one's data exists: the ability to identify models that give the data clear context and impact. If a new piece of data invalidates a widely-accepted model, this can be cited in publications. **Fourth**, one can compare new data against existing data by simply creating a *data-derived model* that directly generates "predictions" by draws from the existing dataset directly. Thus the model validation procedures described in this paper can also be used to perform data validation – that is, check the goodness-of-fit of new data with old data.

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 flexibly express precisely how a model should be interfaced using capabilities. Capabilities can inherit from and call into each other. This is more flexible than the fixed interfaces used in existing modeling competitions or domains like machine learning where only a few interfaces are of interest (e.g. classifiers). **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). In RSA, 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 OCCAM'S RAZOR

All things being equal, simpler models are better. Model complexity, though fundamentally a subjective measure, has many correlates (**McCabe**, 1976), including: 1) model length; 2) memory use; 3) CPU load; 4) # of capabilities. Future *SciUnit* tools will analyze these factors, and may include tools for gathering community ratings about subjective factors like complexity. As a result of these enhancements, scientists will be able to visualize the model validity vs complexity tradeoff in tabular form (e.g. as a column in Figure 6), or as a scatter plot, with the "best" models being in the high validity / low complexity corner of the plot. Validity may be equated to cross-validation accuracy, or the inverse of some loss function. 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 (e.g. as used in the symbolic regression package Eureqa (**Schmidt and Lipson**, 2009)).

5.7 EXPANSION INTO OTHER AREAS OF BIOLOGY

After covering neurophysiology with continued development of *NeuronUnit*, 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.

5.8 PROJECT DEVELOPMENT

Gewaltig and Cannon describe the maturity of a computational software project (**Gewaltig and Cannon**, 2014) as "review ready", "research ready", or "user ready". The development of the underlying SciUnit framework is complete (**Omar et al.**, 2014), and it provides the full interface necessary for continued development of domain-specific unit-testing frameworks – it is research ready. NeuronUnit is "review ready" – we invite others to participate in building it out to satisfy the use cases they encounter. Corresponding test suite repositories on GitHub, containing NeuronUnit-based tests, do not even reach this stage. With the exception of the repositories in development for the QSNMC, and for the Open Source Brain pipeline describe in Section 4, they are non-existent. We plan to continue creating such repositories; more importantly we invite interested parties to create their own, either to guide in-house model development or to assess larger pools of established models.

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

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

Bower, J. 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)

Donoho, D., Maleki, A., Rahman, I., Shahram, M., and Stodden, V. (2008), 15 years of reproducible research in computational harmonic analysis, *IEEE Computing in Science and Engineering*, 11, 818, doi:10.1109/mcse.2009.15

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 Cannon, R. (2014), Current practice in software development for computational neuroscience and how to improve it, *PLoS Comput Biol*, 10, 1, e1003376, doi:10.1371/journal.pcbi.1003376

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

Larson, S. D. and Martone, M. E. (2013), NeuroLex.org: an online framework for neuroscience knowledge, *Frontiers in Neuroinformatics*, 7, 18, doi:10.3389/fninf.2013.00018

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

Migliore, M., Ferrante, M., and Ascoli, G. A. (2005), Signal propagation in oblique dendrites of ca1 pyramidal cells, *Journal of Neurophysiology*, 94, 6, 4145–4155, doi:10.1152/jn.00521.2005

Omar, C., Aldrich, J., and Gerkin, R. C. (2014), Collaborative infrastructure for test-driven scientic model validation, *ICSE: New Ideas and Emerging Results*, Accepted, 1–4

Pérez, F. and Granger, B. E. (2007), IPython: a System for Interactive Scientific Computing, *Comput. Sci. Eng.*, 9, 3, 21–29

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*