1 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 a contemporary understanding of a model's strengths and weaknesses, to adequately compare competing models, to determine the state-of-the-art, and to precisely identify open modeling problems.

Software engineers validate software by developing simple executable tests, called "unit tests", that individually check a portion of a program against a single correctness criterion. Suites of such unit tests that collectively capture the desired program behavior 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 single empirical observations 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. Building on this framework, we develop *NeuronUnit*, a library that supports validation testing for neurophysiology.

`is this accurate?`

2 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. Unfortunately, in many areas, the number of publications being generated every year can overwhelm even the most conscientious scientists .

`citation/stats?`

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 software project and validate it by measuring how well each component achieves its stated goals. 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 collectively specify the scope of each component and validate the functionality of its implementation as it is being developed and modified. 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 [**?**].

`cite tdd`

Test-driven methodologies have started to see success in science 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)[1] investigates the complexity-accuracy tradeoff among reduced models of excitable membranes; the "Hopfield" challenge[2] tested techniques for generating neuronal network form
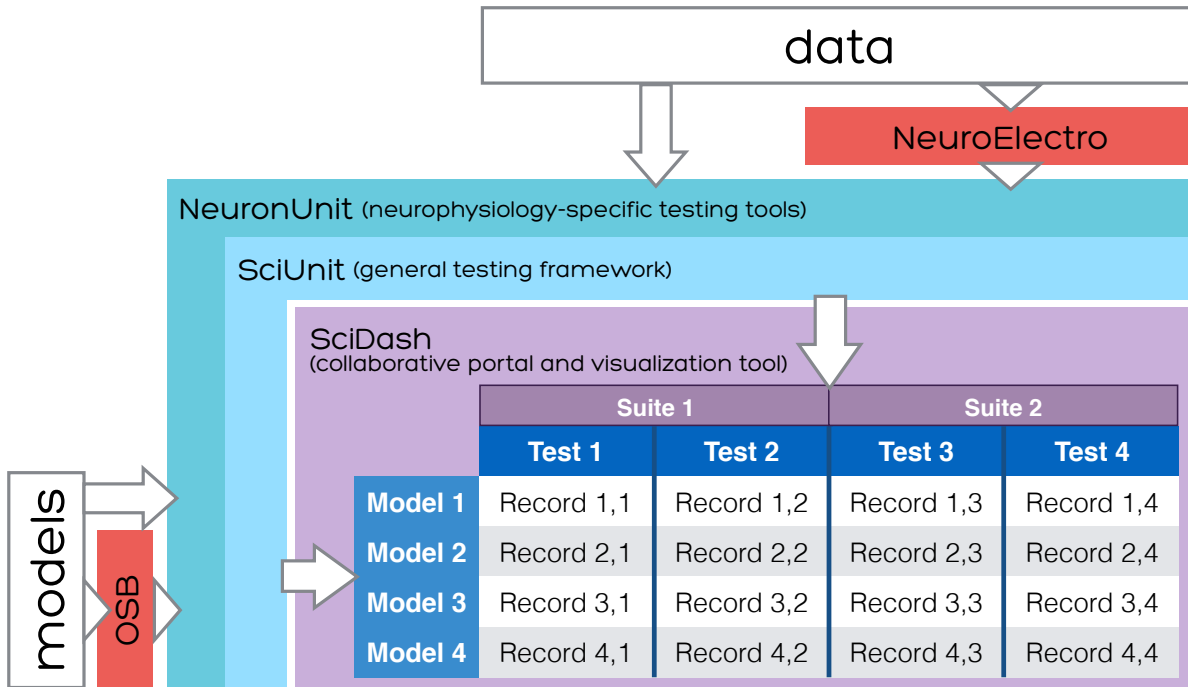
Figure 1: Paper overview.

given its function; the Neural Prediction Challenge sought the best stimulus reconstructions, given neuronal activity[3]; the Diadem challenge is advancing the art of neurite reconstruction[4]; and examples from other subfields of biology abound[5].

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 have developed a discipline-agnostic framework for developing scientific validation test suites called *SciUnit*[6]. To demonstrate the power of this framework and show how it can be used to bridge several existing neuroinformatics tools, we then developed *NeuronUnit*. This library allows scientists to build *SciUnit* tests that validate NeuroML models against electrophysiological data collected by the *NeuroElectro Project* [7], leveraging facilities from the *NeuroTools* project to...

3   Validation Testing with *SciUnit*

3.1   Example: Quantitative Single Neuron Modeling Competition   We begin by constructing an example validation test suite. 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 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. In our example, we will determine goodness-of-fit by first calculating a p-value from a chi-squared statistic for each prediction and then combining these p-values using Fisher's method[8].

Alongside this *spike count test*, a number of other tests capturing different features of the data can be specified to produce a more comprehensive suite. For data of this sort, the QSNMC defined 17 other

```python
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 run_test(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: A single neuron spike count test family implemented using *SciUnit*. This family contains logic common to many different systems, so it is distributed in the *NeuronUnit* repository (see Sec. 4.1).

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[1]. 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.

3.2 Implementing a Validation Test in *SciUnit* Fig. 2 shows how a scientist would 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 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:

```python
1    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* [9] (Sec. 4.1).

Classes that implement the `sciunit.Test` interface must contain a `run_test` 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

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

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

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

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

`required_capabilities` attribute, seen on line 12 of Fig. 2. Capabilities are simply collections of methods that a test may need to invoke to receive relevant data, and are analogous to *interfaces* in e.g. Java [10]. In Python, they 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 `run_test` method implements the goodness-of-fit metric described above, returning an instance of `sciunit.PValue`, a subclass of `sciunit.Score` included with *SciUnit*. In addition to the score itself, the returned score object also contains metadata, via the `related_data` parameter, for those who may wish to examine the result in more detail. In this case we save the input currents, the model outputs and the observed means and standard deviations alongside the score (line 24).

3.3 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 needed in the constructor. A particular model is an object created by supplying the parameters.

A simple family of models, LinearModel}, implementing the capability in Fig. \reffig:capabilit as well as another capability shown in Fig. 4, is shown in Fig. 5. Models in this family, `LinearModel`, produce a spike count by applying a linear transformation to the mean of the provided input current. The family is parameterized by a scale factor and offset, both scalars. To create a *particular* model, the modeler may choose particular parameter values, just as with test families:

```
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 to the one in Fig. 2 would add a second required capability (Fig. 4) and a corresponding training phase to the `run_test` method. This design would be relevant only 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 modeler. Although Fig. 2 does not include a training phase, if training data is available then models that nevertheless implement a training capability (like `LinearModel`) can simply be trained explicitly by calling the capability method just like any other Python method:

```
1  CA1_linear_model_fit = LinearModel()
2  CA1_linear_model_fit.train_with_currents(CA1_training_in, CA1_training_out)
```

Finally, a test is executed against a model instance, using the `sciunit.judge` function:

```
1  score = sciunit.judge(CA1_sc_test, CA1_linear_model_heuristic)
```

Collection of tests with mutually coherent requirments, i.e. various summaries of the available empirical data, can form test suites. The following is a test suite that could be used for a simplified version of the QSNMC:

```
1  class LinearModel(sciunit.Model, SpikeCountFromCurrent,
2      TrainSpikeCountFromCurrent):
3    def __init__(self, scale=None, offset=None):
4      self.scale, self.offset = scale, offset
5
6    def spike_count_from_current(self, input):
7      return int(self.scale*numpy.mean(input) + self.offset)
8
9    def train_with_currents(self, currents, counts):
10     means = [numpy.mean(c) for c in currents]
11     [self.offset, self.scale] = numpy.polyfit(means, counts, deg=1)
```

Figure 5: A simple model that returns a spike count by scaling the mean of the input by a fixed parameter.

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

When a test suite is executed against a model, it produces summary data that can be shown on the console or visualized by other tools, such as the web application described in Sec. 4.2.

4   Results

4.1   *NeuronUnit*: A Suite of Tests for Neurophysiology By using machine-readable models from Open Source Brain (OSB), and machine-readable data from resources like The NeuroElectro Project[7], we have produced a suite of data-driven validation tests in *NeuronUnit*, as well as tools for capability implementation, and model interfaces for a representative set of canonical neurophysiology models, all publicly available.

Here we describe standard neuroinformatics tools we have adopted to develop *NeuronUnit*[9].

4.1.1   Models from NeuroML NeuroML is a standardized model description language for neuroscience [11]. 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[12, 13] 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).

The Open Source Brain project (OSB,[14]) curates many models described in NeuroML. OSB-curated projects are converted from their native format into NeuroML, and run on major neural simulators[15–18]. 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[19] 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. Working together with OSB is part of our **first collaboration**, and our integration efforts can be publicly tracked[20].

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

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

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

for establishing *how well* they work, i.e. how well the models accord with data. *NeuroUnit* 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.

4.1.2   Capabilities from NeuroTools NeuroTools[21] 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`.

4.1.3   Reference Data for Tests from NeuroElectro Answering *how well* requires validation testing against data. The NeuroElectro project[7] is an effort to curate all published single cell neurophysiology data[22]. 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. Continued development of the NeuroElectro API, through which data are systematically exposed to test authors, represents our **second collaboration**[23].

4.2   *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* [24] 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 [25] stored on GitHub [26]. *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

6

Results for suite Single Neuron Suite B

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

```
 1  # Interface with neuroelectro.org to search for spike widths of CA1 Pyramidal cells.
 2  reference_data = NeuroElectroSummary(neuron={'id':85}, ephysprop={'id':23})
 3  reference_data.get_values()  # Get summary data for the above.
 4  model = CA1PyramidalCellModel(population_name="CG_CML_0") # Initialize the model
        with some parameters.
 5  test = SpikeWidthTestDynamic( # Initialize the test.
 6      reference_data = {'mean':reference_data.mean, 'std':reference_data.std}, #
            Summary statistics from the reference data
 7      model_args = {'current':40.0}, # Somatic current injection in pA.
 8      comparator = ZComparator), # A comparison class that implements a Z-score.
 9  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.
10  result.summarize() # Summarize the result.
```

Figure 9: Working example of a testing in *NeuronUnit*

interest to a particular research community. The **third layer** is a "dashboard" view of the state of a field, visible 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

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. 4.2).

integrate this somewhere down here

5.1 A Complete Pipeline Although the tools described in Sec. 4.1 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[9] 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. As part of our outreach efforts (Sec. **??**) we will train modelers in the use of this framework, and encourage them to use it as part of their workflow of choice. 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[11], 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[15], this functionality is already mature and in use. This minimizes modeler effort for a large and growing number of models. **Third**, modelers have an incentive to demonstrate publicly their models' validity. Participation in public modeling competitions (Sec. **??**) 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 ca-

pability 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[27, 28], a general systems biology markup language. **Third**, cross-level testing can use "Representational Similarity Analysis" (RSA)[29], 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[30–33], and is accelerated by an open framework. The 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.

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[29]. 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[34], 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 **??**), 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[35].

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[36]), neuroanatomy (NeuroHDF,[37]) and other sub-disciplines. SBML[27, 28] 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[38], which through open, collaborative development seeks to model the entire organism C. elegans.

6 Acknowledgements

References

[1] Renaud Jolivet, Felix Schürmann, Thomas Berger, Richard Naud, Wulfram Gerstner, and Arnd Roth. The quantitative single-neuron modeling competition. *Biological Cybernetics*, 99(4):417–426, November 2008.

[2] J J Hopfield and C D Brody. 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, December 2000. PMID: 11095747.

[3] *http://neuralprediction.berkeley.edu/*.

[4] *http://www.diademchallenge.org*.

[5] *http://www.the-dream-project.org/*.

[6] *http://github.com/scidash/sciunit*.

[7] *http://www.neuroelectro.org/*.

[8] Ronald A Fisher. *Statistical Methods for Research Workers*. Oliver and Boyd, Edinburgh, 1925.

[9] *http://github.com/scidash/neurounit*.

[10] *http://docs.oracle.com/javase/tutorial/java/concepts/interface.html*.

[11] Padraig Gleeson, Sharon Crook, Robert C Cannon, Michael L Hines, Guy O Billings, Matteo Farinella, Thomas M Morse, Andrew P Davison, Subhasis Ray, Upinder S Bhalla, Simon R Barnes, Yoana D Dimitrova, and R Angus Silver. 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, June 2010. PMID: 20585541.

[12] *http://www.neuroconstruct.org*.

[13] Padraig Gleeson, Volker Steuber, and R Angus Silver. neuroConstruct: a tool for modeling networks of neurons in 3D space. *Neuron*, 54(2):219–235, April 2007. PMID: 17442244.

[14] *http://www.opensourcebrain.org*.

[15] *http://www.neuron.yale.edu/neuron*.

[16] *http://genesis-sim.org/*.

[17] *http://www.nest-initiative.org/*.

[18] *http://moose.ncbs.res.in/*.

[19] *http://opensourcebrain.org/projects/ca1pyramidalcell*.

[20] *http://github.com/rgerkin/neuroConstruct*.

[21] *http://neuralensemble.org*.

[22] SJ Tripathy, J Saviskaya, RC Gerkin, and NN Urban. NeuroElectro: a database describing the electro-physiology properties of different neuron types. *Neuroinformatics Meeting*, 2012.

[23] *http://bitbucket.org/rgerkin/neuroelectro_org*.

[24] *http://www.scidash.org*.

[25] Karthik Ram. Git can facilitate greater reproducibility and increased transparency in science. *Source code for biology and medicine*, 8(1):7, 2013. PMID: 23448176.

[26] *http://www.neuron.yale.edu/neuron*.

[27] *http://sbml.org/*.

[28] M Hucka, A Finney, H M Sauro, H Bolouri, J C Doyle, H Kitano, A P Arkin, B J Bornstein, D Bray, A Cornish-Bowden, A A Cuellar, S Dronov, E D Gilles, M Ginkel, V Gor, I I Goryanin, W J Hedley, T C Hodgman, J-H Hofmeyr, P J Hunter, N S Juty, J L Kasberger, A Kremling, U Kummer, N Le Novère, L M Loew, D Lucio, P Mendes, E Minch, E D Mjolsness, Y Nakayama, M R Nelson, P F Nielsen, T Sakurada, J C Schaff, B E Shapiro, T S Shimizu, H D Spence, J Stelling, K Takahashi, M Tomita, J Wagner, J Wang, and SBML Forum. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics (Oxford, England)*, 19(4):524–531, March 2003. PMID: 12611808.

[29] Nikolaus Kriegeskorte, Marieke Mur, and Peter Bandettini. Representational similarity analysis - connecting the branches of systems neuroscience. *Frontiers in systems neuroscience*, 2:4, 2008. PMID: 19104670.

[30] Katherine S Button, John P A Ioannidis, Claire Mokrysz, Brian A Nosek, Jonathan Flint, Emma S J Robinson, and Marcus R Munafò. Power failure: why small sample size undermines the reliability of neuroscience. *Nature reviews. Neuroscience*, 14(5):365–376, May 2013. PMID: 23571845.

[31] Nikolaus Kriegeskorte, W Kyle Simmons, Patrick S F Bellgowan, and Chris I Baker. Circular analysis in systems neuroscience: the dangers of double dipping. *Nature neuroscience*, 12(5):535–540, May 2009. PMID: 19396166.

[32] Sally Galbraith, James A. Daniel, and Bryce Vissel. A study of clustered data and approaches to its analysis. *The Journal of Neuroscience*, 30(32):10601–10608, August 2010. PMID: 20702692.

[33] *http://prefrontal.org/files/posters/Bennett-Salmon-2009.jpg/*.

[34] TJ McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, 1976.

[35] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science (New York, N.Y.)*, 324(5923):81–85, April 2009. PMID: 19342586.

[36] *http://nipy.org/nibabel*.

[37] *http://neurohdf.readthedocs.org*.

[38] *http://www.openworm.org/*.