

Data-driven validation of neuroscience models

Richard (Rick) Gerkin, PhD
Arizona State University, USA

MODEL VALIDATION

MODEL VALIDATION

- A scientific model has high *validity* if:

MODEL VALIDATION

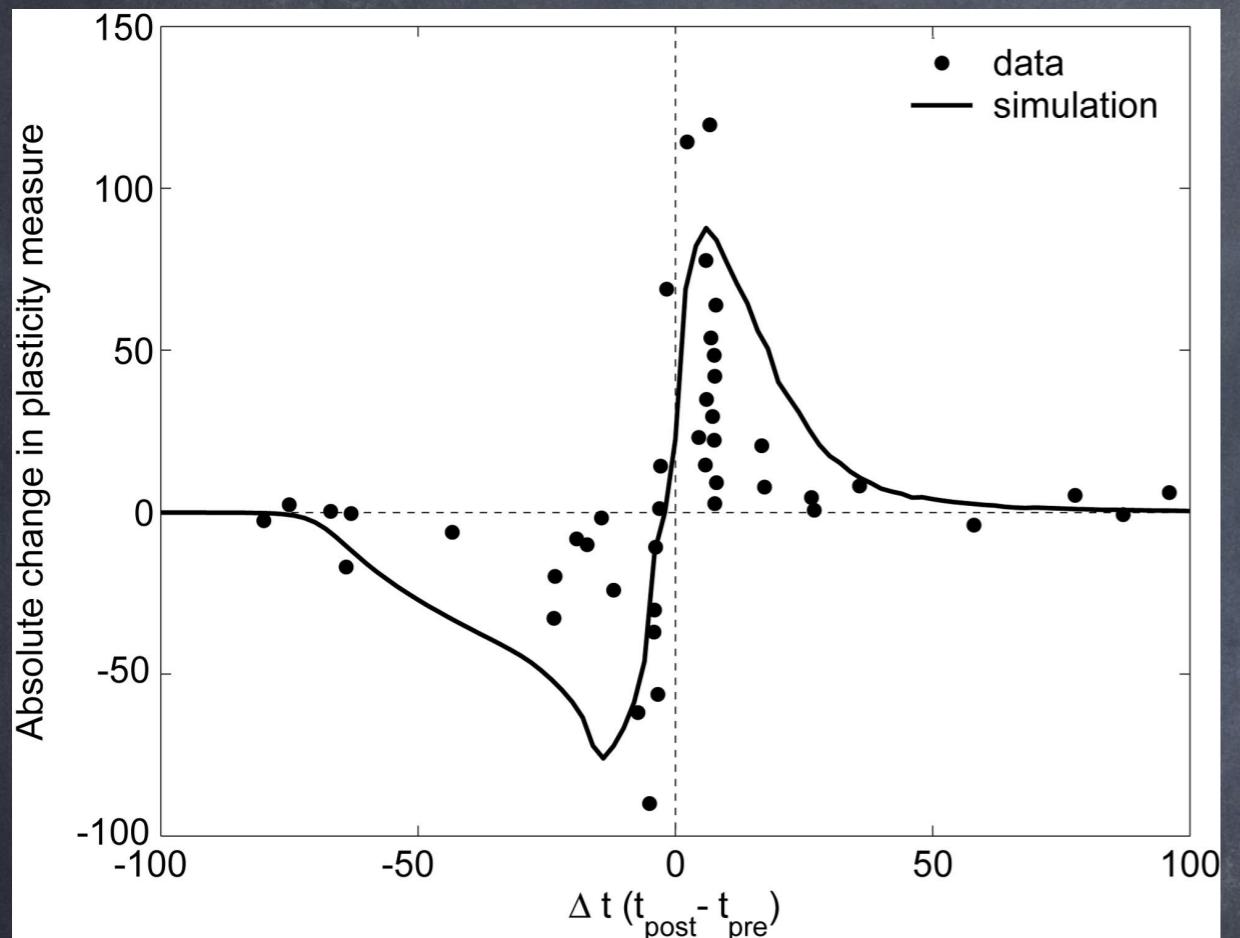
- A scientific model has high *validity* if:
 - It is *consistent with* a wide range of previously gathered data.

MODEL VALIDATION

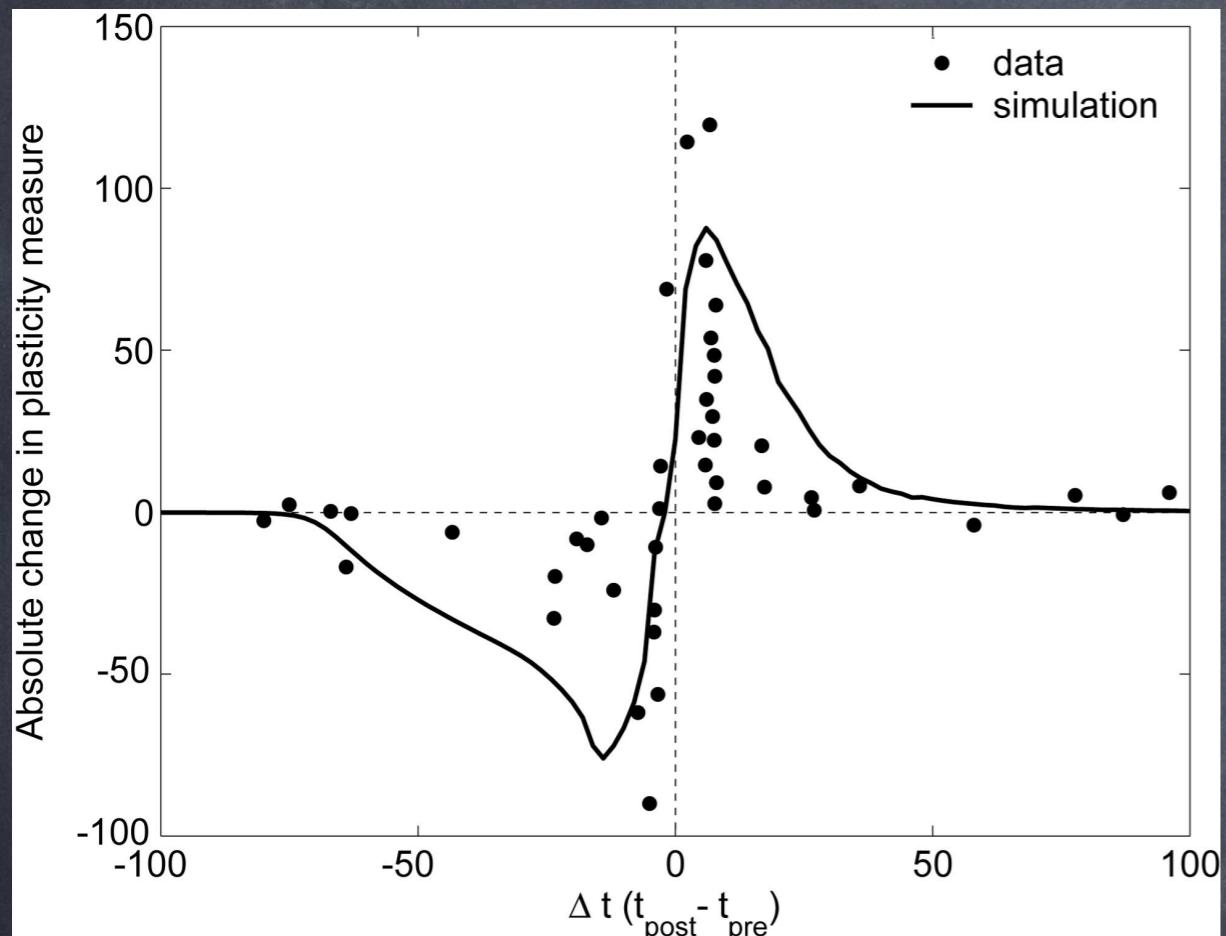
- A scientific model has high *validity* if:
 - It is *consistent with* a wide range of previously gathered data.
 - It can *predict* the results of many future experiments.

TESTING VALIDITY

TESTING VALIDITY

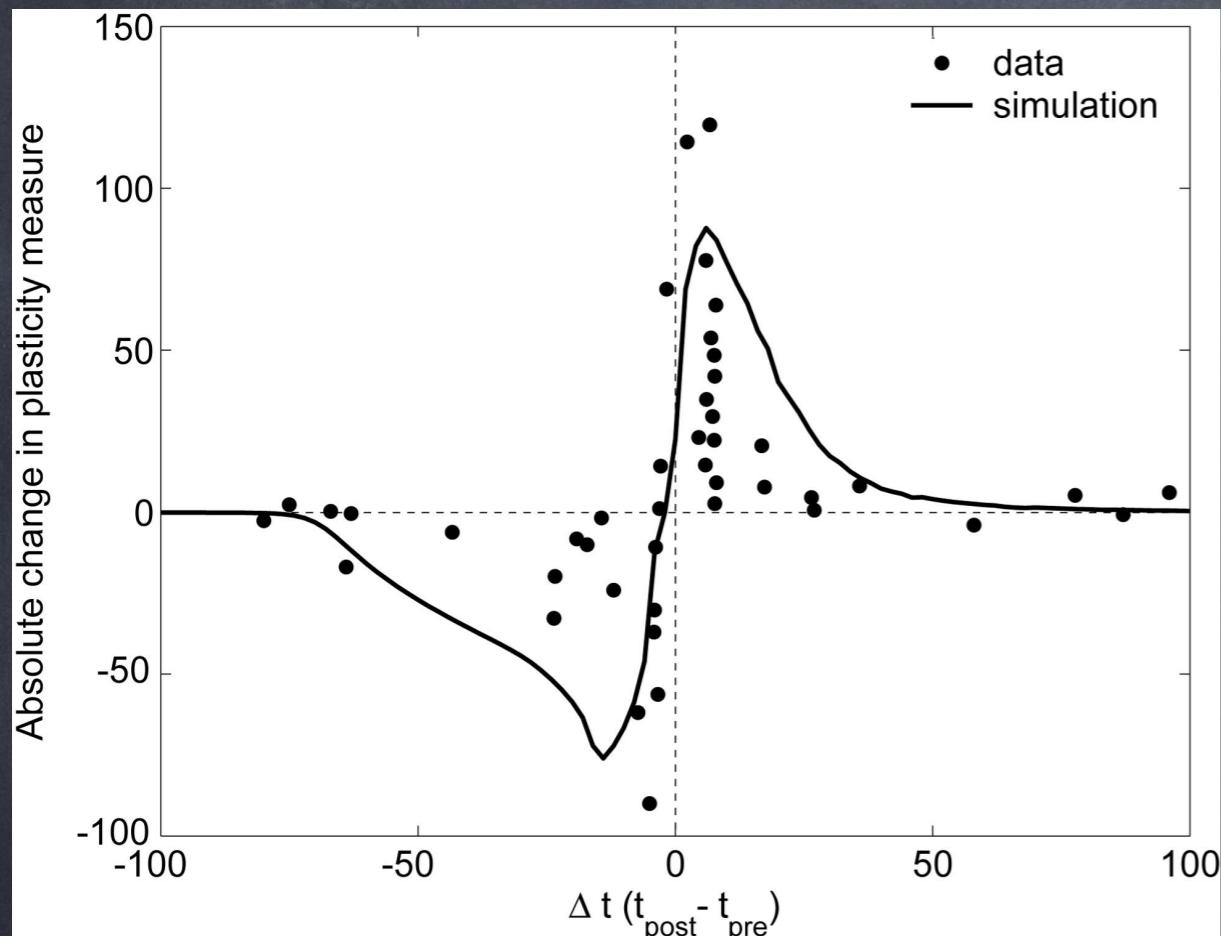


TESTING VALIDITY



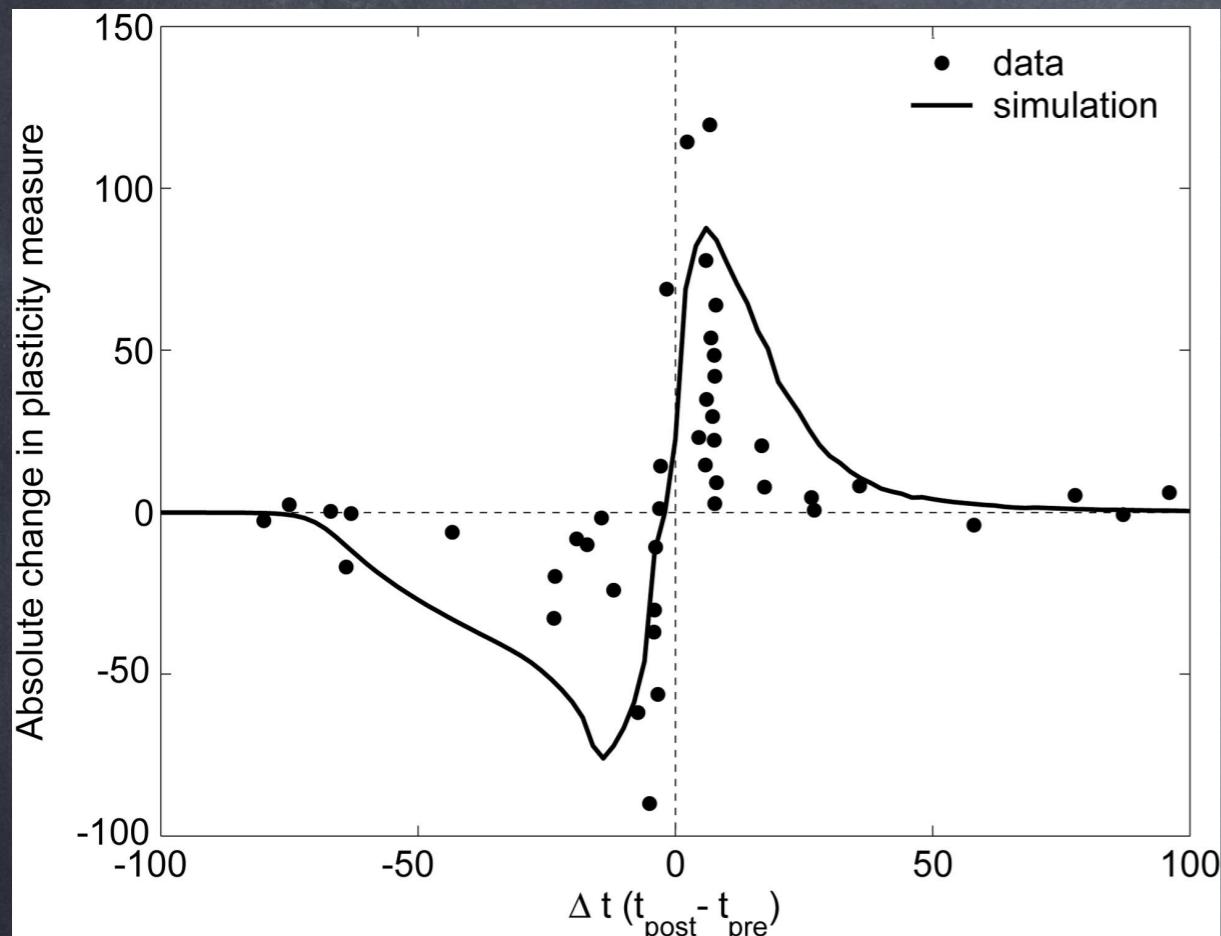
- Most claims about model validity are informal.

TESTING VALIDITY



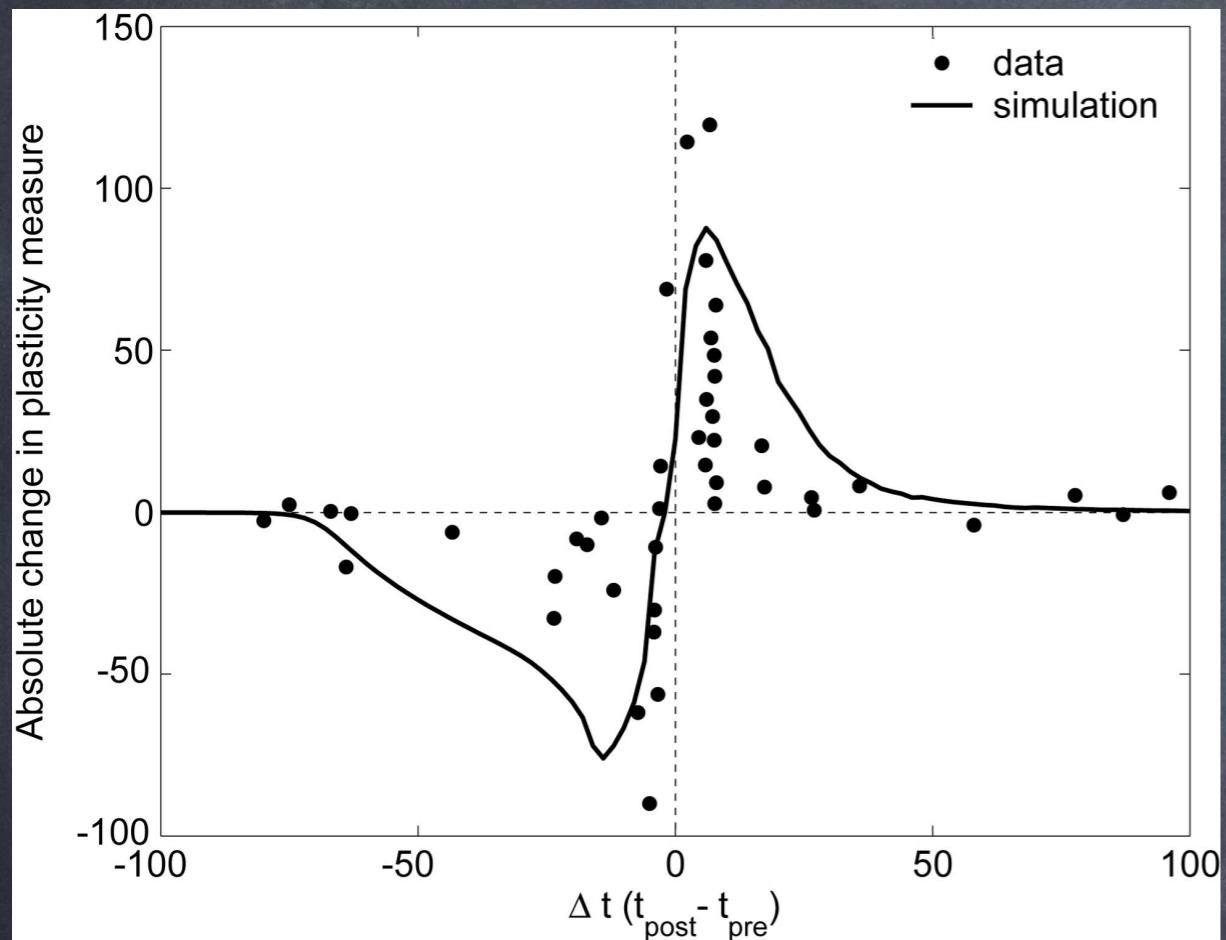
- Most claims about model validity are informal.
- These claims cannot be formally evaluated.

TESTING VALIDITY



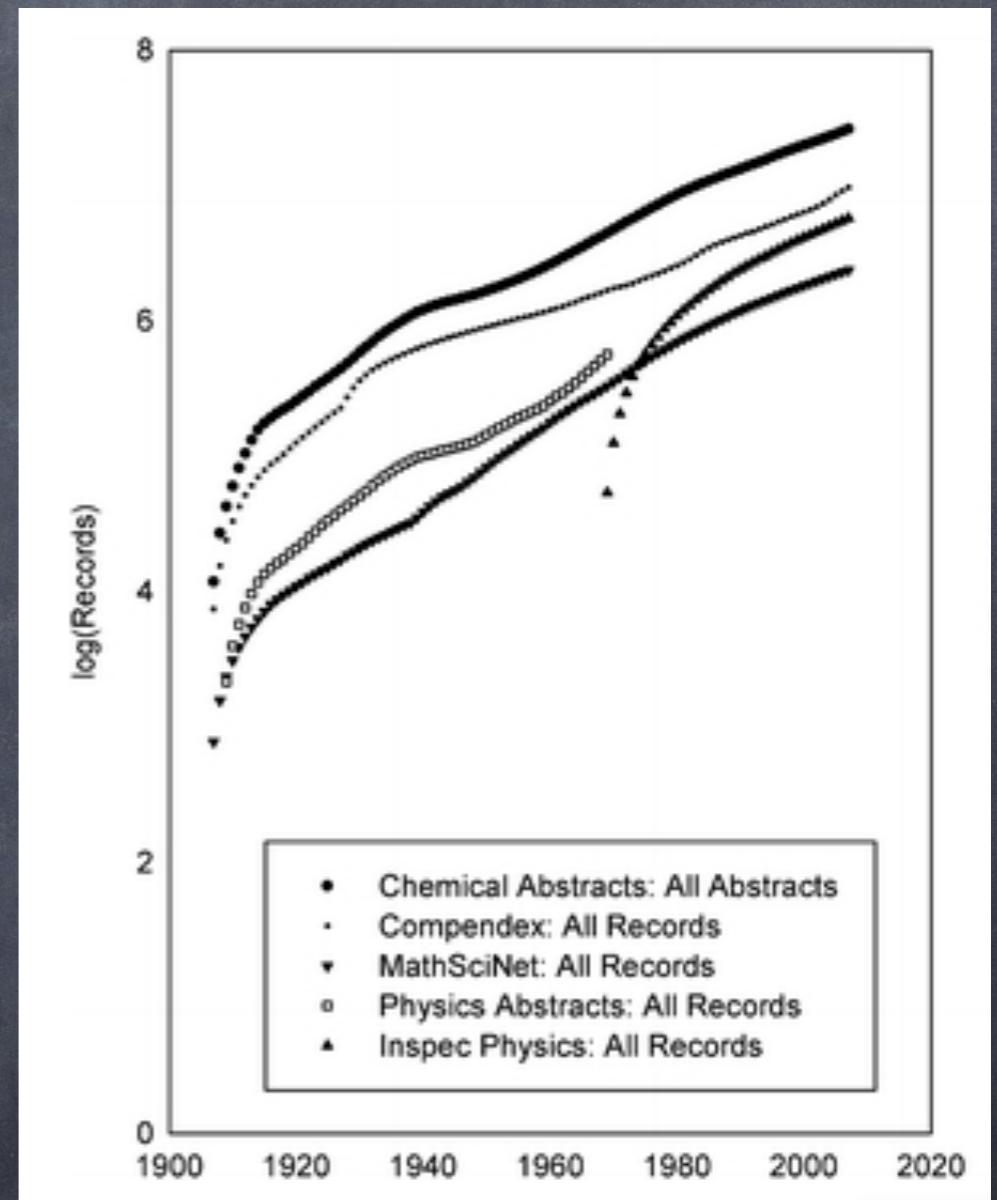
- Most claims about model validity are informal.
- These claims cannot be formally evaluated.
- Claims are difficult to locate.

TESTING VALIDITY

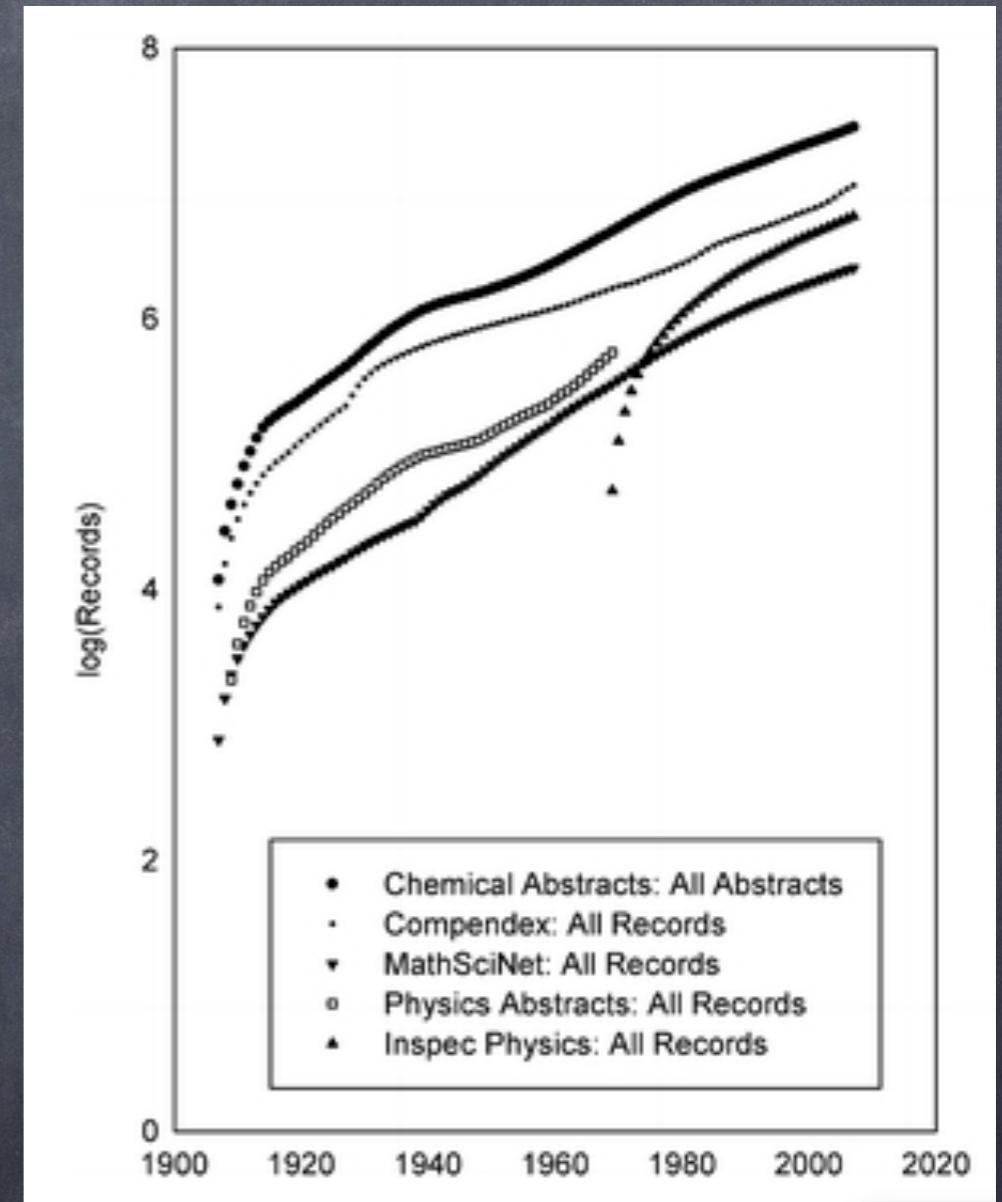


- Most claims about model validity are informal.
- These claims cannot be formally evaluated.
- Claims are difficult to locate.
- Falsifiable claims spanning many data sets are rare.

TESTING VALIDITY



TESTING VALIDITY



Huge amounts of data!
Even *informal validation* is becoming difficult.

TESTING VALIDITY

TESTING VALIDITY

- Not testing models rigorously leaves a field with an **unclear** view of:

TESTING VALIDITY

- Not testing models rigorously leaves a field with an **unclear** view of:
 - What has already been explained.

TESTING VALIDITY

- Not testing models rigorously leaves a field with an **unclear** view of:
 - What has already been explained.
 - What needs explaining.

TESTING VALIDITY

- Not testing models rigorously leaves a field with an **unclear** view of:
 - What has already been explained.
 - What needs explaining.
- Needed: a **framework** for validating scientific models, based on established techniques for **formally validating** software.

THE UNIT TEST

THE UNIT TEST

- “unit testing is a software testing method by which individual units of source code ... are tested to determine if they are **fit for use.**”

THE UNIT TEST

- “unit testing is a software testing method by which individual units of source code ... are tested to determine if they are **fit for use.**”
- “one can view a unit as the **smallest testable part** of an application.”

THE UNIT TEST

- “unit testing is a software testing method by which individual units of source code ... are tested to determine if they are **fit for use.**”
- “one can view a unit as the **smallest testable part** of an application.”
- “A unit test provides a **strict, written contract** that the piece of code must satisfy.”

SCIENTIFIC UNIT TESTS

SCIENTIFIC UNIT TESTS

- A **scientific model is just a function** that takes *observations* as inputs and produces *predictions* as output.

SCIENTIFIC UNIT TESTS

- A **scientific model is just a function** that takes *observations* as inputs and produces *predictions* as output.
- The observations include metadata about how they were obtained, to direct the model.

SCIENTIFIC UNIT TESTS

- A **scientific model is just a function** that takes *observations* as inputs and produces *predictions* as output.
- The observations include metadata about how they were obtained, to direct the model.
- Example: A *complete* model of a given neuron type should produce spikes that, given a specified description of the stimulus, **match physiologically observed features** such as spike shapes, rates, interval distributions, etc.

SCIENTIFIC UNIT TESTS

- A **scientific model is just a function** that takes *observations* as inputs and produces *predictions* as output.
 - The observations include metadata about how they were obtained, to direct the model.
- Example: A *complete* model of a given neuron type should produce spikes that, given a specified description of the stimulus, **match physiologically observed features** such as spike shapes, rates, interval distributions, etc.
 - Each such feature can be encoded in a unit test.

SCIUNIT

SCIUNIT

- Showing a match to empirical data in a journal article figure is nice, but not sufficient.

SCIUNIT

- Showing a match to empirical data in a journal article figure is nice, but not sufficient.
- What if we built a **collaborative collection** of *empirically-informed validation tests* and **characterized models** by the collection of **tests that they pass?**

SCIUNIT

- Showing a match to empirical data in a journal article figure is nice, but not sufficient.
- What if we built a **collaborative collection** of *empirically-informed validation tests* and **characterized models** by the collection of **tests that they pass?**
- <http://github.com/scidash/sciunit>

VISUALIZING VALIDITY

VISUALIZING VALIDITY

JOURNAL OF GEOPHYSICAL RESEARCH, VOL. 104, NO. A10, PAGES 22,375–22,388, OCTOBER 1, 1999

A synthesis of solar cycle prediction techniques

David H. Hathaway, Robert M. Wilson, and Edwin J. Reichmann

NASA Marshall Space Flight Center, Huntsville, Alabama

Table 3. Precursor Prediction Method Errors (Prediction - Observed) for Cycles 19-22

Prediction Method	19	20	21	22	RMS
Ohl's method	-55.4	19.1	21.8	4.4	31.3
Feynman's method	-42.8	9.6	26.9	3.6	25.8
Thompson's method	-17.8	8.7	-26.5	-13.6	17.9

VISUALIZING VALIDITY

JOURNAL OF GEOPHYSICAL RESEARCH, VOL. 104, NO. A10, PAGES 22,375–22,388, OCTOBER 1, 1999

A synthesis of solar cycle prediction techniques

David H. Hathaway, Robert M. Wilson, and Edwin J. Reichmann

NASA Marshall Space Flight Center, Huntsville, Alabama

Table 3. Precursor Prediction Method Errors (Prediction - Observed) for Cycles 19-22

Prediction Method	19	20	21	22	RMS
Ohl's method	-55.4	19.1	21.8	4.4	31.3
Feynman's method	-42.8	9.6	26.9	3.6	25.8
Thompson's method	-17.8	8.7	-26.5	-13.6	17.9

Models

VISUALIZING VALIDITY

JOURNAL OF GEOPHYSICAL RESEARCH, VOL. 104, NO. A10, PAGES 22,375–22,388, OCTOBER 1, 1999

A synthesis of solar cycle prediction techniques

David H. Hathaway, Robert M. Wilson, and Edwin J. Reichmann

NASA Marshall Space Flight Center, Huntsville, Alabama

Tests

Table 3. Precursor Prediction Method Errors (Prediction - Observed) for Cycles 19-22

Prediction Method	19	20	21	22	RMS
Ohl's method	-55.4	19.1	21.8	4.4	31.3
Feynman's method	-42.8	9.6	26.9	3.6	25.8
Thompson's method	-17.8	8.7	-26.5	-13.6	17.9

Models

VISUALIZING VALIDITY

JOURNAL OF GEOPHYSICAL RESEARCH, VOL. 104, NO. A10, PAGES 22,375–22,388, OCTOBER 1, 1999

A synthesis of solar cycle prediction techniques

David H. Hathaway, Robert M. Wilson, and Edwin J. Reichmann
NASA Marshall Space Flight Center, Huntsville, Alabama

Tests

Table 3. Precursor Prediction Method Errors (Prediction - Observed) for Cycles 19-22

Prediction Method	19	20	21	22	RMS
Ohl's method	-55.4	19.1	21.8	4.4	31.3
Feynman's method	-42.8	9.6	26.9	3.6	25.8
Thompson's method	-17.8	8.7	-26.5	-13.6	17.9

Models

Goodness-of-Fit

VISUALIZING VALIDITY

JOURNAL OF GEOPHYSICAL RESEARCH, VOL. 104, NO. A10, PAGES 22,375–22,388, OCTOBER 1, 1999

A synthesis of solar cycle prediction techniques

David H. Hathaway, Robert M. Wilson, and Edwin J. Reichmann
NASA Marshall Space Flight Center, Huntsville, Alabama

Tests

Table 3. Precursor Prediction Method Errors (Prediction - Observed) for Cycles 19-22

Prediction Method	19	20	21	22	RMS
Ohl's method	-55.4	19.1	21.8	4.4	31.3
Feynman's method	-42.8	9.6	26.9	3.6	25.8
Thompson's method	-17.8	8.7	-26.5	-13.6	17.9

Test Suite

Models

Goodness-of-Fit

SCIUNIT: CHALLENGES

SCIUNIT: CHALLENGES

- How to *interface* with a wide range of model scales, languages, and goals?

SCIUNIT: CHALLENGES

- How to *interface* with a wide range of model scales, languages, and goals?
- How to *minimize development time* for writing model validation tests?

SCIUNIT: CHALLENGES

- How to *interface* with a wide range of model scales, languages, and goals?
- How to *minimize development time* for writing model validation tests?
- How to adjudicate whether a test *score* really captures the functionality it claims to test?

SCIUNIT: CHALLENGES

- How to *interface* with a wide range of model scales, languages, and goals?
- How to *minimize development time* for writing model validation tests?
- How to adjudicate whether a test *score* really captures the functionality it claims to test?
- We aim to solve these with:

SCIUNIT: CHALLENGES

- How to *interface* with a wide range of model scales, languages, and goals?
- How to *minimize development time* for writing model validation tests?
- How to adjudicate whether a test *score* really captures the functionality it claims to test?
- We aim to solve these with:
 - *unit testing philosophy*

SCIUNIT: CHALLENGES

- How to *interface* with a wide range of model scales, languages, and goals?
- How to *minimize development time* for writing model validation tests?
- How to adjudicate whether a test *score* really captures the functionality it claims to test?
- We aim to solve these with:
 - *unit testing philosophy*
 - *domain standards and tools*

SCIUNIT: CHALLENGES

- How to *interface* with a wide range of model scales, languages, and goals?
- How to *minimize development time* for writing model validation tests?
- How to adjudicate whether a test *score* really captures the functionality it claims to test?
- We aim to solve these with:
 - *unit testing philosophy*
 - *domain standards and tools*
 - *collaborative development*

SCIUNIT: CHALLENGES

- How to ***interface*** with a wide range of model types, languages, and goals?
- How to ***minimize development time*** for writing model validation tests?
- How to adjudicate whether a test ***score*** really captures the functionality it claims to test?
- We aim to solve these with:
 - ***unit testing philosophy***
 - ***domain standards and tools***
 - ***collaborative development***

SCIUNIT: CAPABILITIES

SCIUNIT: CAPABILITIES

Test

I need you to have a soma, to receive somatic current injection, and to produce action potentials... can you do that?

SCIUNIT: CAPABILITIES

Test

I need you to have a soma, to receive somatic current injection, and to produce action potentials... can you do that?

Model1

Sorry, I'm a non-spiking model.
I cannot produce action potentials.

SCIUNIT: CAPABILITIES

Test

I need you to have a soma, to receive somatic current injection, and to produce action potentials... can you do that?

Model1

Sorry, I'm a non-spiking model.
I cannot produce action potentials.

Model2

I can do all those things.

SCIUNIT: CAPABILITIES

Test

I need you to have a soma, to receive somatic current injection, and to produce action potentials... can you do that?

Model1

Sorry, I'm a non-spiking model.
I cannot produce action potentials.

Model2

I can do all those things.

Test

Model2, bring it on!

SCIUNIT: CHALLENGES

- How to *interface* with a wide range of model types, languages, and goals?
- How to ***minimize development time*** for writing model validation tests?
- How to adjudicate whether a test *score* really captures the functionality it claims to test?
- We aim to solve these with:
 - *unit testing philosophy*
 - ***domain standards and tools***
 - *collaborative development*

NEUROELECTRO

<http://www.neuroelectro.org>

Published literature

Novel subcellular distribution pattern of A-type K⁺ channels on neuronal surface.

Unique clustering of A-type potassium channels on different cell types of the main olfactory bulb.

Kollo M, Holderith N, Antal M, Nusser Z.
Theoretical and functional studies predicted a highly non-uniform distribution of voltage-gated ion channels on the neuronal surface. This was confirmed by recent immunolocalization experiments for Na⁺, Ca²⁺, hyperpolarization activated mixed cation and K⁺ channels. These experiments also indicated that some K⁺ channels were clustered in synaptic or non-synaptic membrane specializations. Here we analysed the subcellular distribution of Kv4.2 and Kv4.3 subunits in the rat main olfactory bulb at high resolution to address whether clustering characterizes their distribution, and whether they are concentrated in synaptic or non-synaptic junctions. The cell surface distribution of the Kv4.2 and Kv4.3 subunits is highly non-uniform. Strong Kv4.2 subunit-immunopositive clusters were detected in intercellular junctions made by mitral, external tufted and granule cells (GCs). We also found Kv4.3 subunit-immunopositive clusters in periglomerular (PGC), deep short-axon and GCs. In the juxtaglomerular region some calretinin-immunopositive glial cells enwrap neighboring PGC somata in a cap-like manner. Kv4.3 subunit clusters are present in the cap membrane that directly contacts the PGC, but not the one that faces the neuropil. In membrane specializations established by members of the same cell type, K⁺ channels are enriched in both membranes, whereas specializations between different cell types contain a high density of channels asymmetrically. None of the K⁺ channel-rich junctions showed any of the ultrastructural features of known chemical synapses. Our study provides evidence for highly non-uniform subcellular distributions of A-type K⁺ channels and predicts their involvements in novel

Physiology database

Olfactory Bulb Mitral Cell

Input resistance	200 MΩ
V _{rest}	-65 mV
Spike width	1 ms
...	

CA1 Pyramidal Cell

Input resistance	400 MΩ
V _{rest}	-70 mV
Spike width	.5 ms
...	

Extracted from Literature

NEUROELECTRO

```
In [1]: from neuronunit.neuroelectro import NeuroElectroSummary
```

```
In [2]: summary = NeuroElectroSummary(neuron={'name':'Hippocampus CA1 Pyramidal Cell'},  
...: ephysprop={'name':'spike width'})
```

```
In [3]: observation = summary.get_observation(show=True)
```

Getting data values from neuroelectro.org

http://www.neuroelectro.org/api/1/nes/?e__name=spike+width&n__name=Hippocampus+CA1+Pyramidal+Cell

```
{u'e': {u'definition': u'Duration of AP, not explicitly referred to as half-width',  
u'id': 23,  
u'name': u'spike width',  
u'nlex_id': None,  
u'norm_criteria': u'Values are unchanged from those reported. Values currently lump multiple measures of spike width which do not  
explicitly denote spike half-width. Refer to individual articles for definition and calculation methodology.'},  
u'n': {u'id': 85,  
u'name': u'Hippocampus CA1 pyramidal cell',  
u'neuron_db_id': 258,  
u'nlex_id': u'sao830368389'},  
u'num_articles': 6,  
u'num_nedms': 10,  
u'value_mean': 2.602083333333333,  
u'value_sd': 0.766070080381394}
```

```
In [4]: from neuronunit.tests import SpikeWidthTest
```

```
In [5]: ca1_pyramidal_spike_width_test = SpikeWidthTest(observation = observation)
```

NEUROELECTRO

```
In [1]: from neuronunit.neuroelectro import NeuroElectroSummary
```

```
In [2]: summary = NeuroElectroSummary(neuron={'name':'Hippocampus CA1 Pyramidal Cell'},  
...: ephysprop={'name':'spike width'})
```

```
In [3]: observation = summary.get_observation(show=True)
```

Getting data values from neuroelectro.org

http://www.neuroelectro.org/api/1/nes/?e__name=spike+width&n__name=Hippocampus+CA1+Pyramidal+Cell

```
{u'e': {u'definition': u'Duration of AP, not explicitly referred to as half-width',  
u'id': 23,  
u'name': u'spike width',  
u'nlex_id': None,  
u'norm_criteria': u'Values are unchanged from those reported. Values currently lump multiple measures of spike width which do not  
explicitly denote spike half-width. Refer to individual articles for definition and calculation methodology.'},  
u'n': {u'id': 85,  
u'name': u'Hippocampus CA1 pyramidal cell',  
u'neuron_db_id': 258,  
u'nlex_id': u'sao830368389'},  
u'num_articles': 6,  
u'num_nedms': 10,  
u'value_mean': 2.602083333333333,  
u'value_sd': 0.766070080381394}
```

```
In [4]: from neuronunit.tests import SpikeWidthTest
```

```
In [5]: ca1_pyramidal_spike_width_test = SpikeWidthTest(observation = observation)
```

- Uses the NeuroElectro API (and NeuroLex) to get collated data about a named *neuron type* for a specified *electrophysiological property* in order to *parameterize a test*.

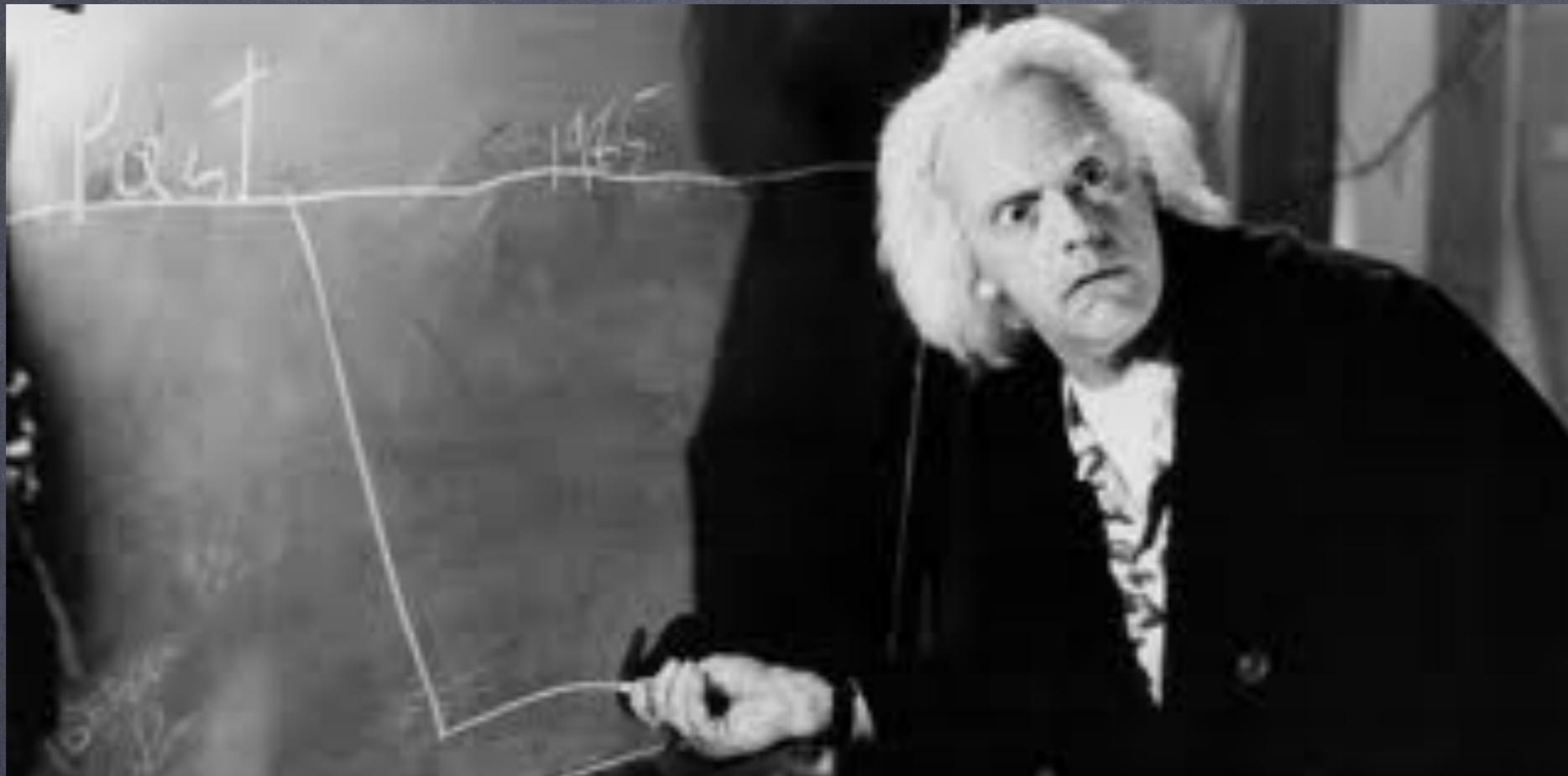
SCIUNIT: CHALLENGES

- How to *interface* with a wide range of model types, languages, and goals?
- How to *minimize development time* for writing model validation tests?
- How to adjudicate whether a test **score** really captures the functionality it claims to test?
- We aim to solve these with:
 - *unit testing philosophy*
 - *domain standards and tools*
 - *collaborative development*

SCIUNIT: TESTS



SCIUNIT: TESTS



Fork it!



COMPETITION

```
from QSNMC.tests import tests
from QSNMC.models import models

for model in models:
    for test in tests:
        score = test.judge(model)
        score.summarize()
```

Quantitative Single-Neuron Modeling: Competition 2009

Richard Naud^{1*}, Thomas Berger¹, Brice Bathellier², Matteo Carandini³ and
Wulfram Gerstner¹

¹ Ecole Polytechnique Federal de Lausanne (EPFL), Switzerland

² University of Bern, Switzerland

³ University College London, United Kingdom

COMPETITION

```
from QSNMC.tests import tests
from QSNMC.models import models

for model in models:
    for test in tests:
        score = test.judge(model)
        score.summarize()
```

Quantitative Single-Neuron Modeling: Competition 2009

Richard Naud^{1*}, Thomas Berger¹, Brice Bathellier², Matteo Carandini³ and
Wulfram Gerstner¹

¹ Ecole Polytechnique Federal de Lausanne (EPFL), Switzerland

² University of Bern, Switzerland

³ University College London, United Kingdom

<https://github.com/incf/qsnmc>

COMPETITION

SciUnit can also provide a common evaluation and visualization framework for competitions between *algorithms or methods*:

- Spike-sorting algorithms
- Spike time extraction from fast calcium imaging.
- Neural system prediction and identification challenge (*nuSPIC*)
- Brain computer interface (BCI, PhyPa (<http://www.phypa.org>)
- Seizure prediction (Sejnowski)

ACTIVE USE CASES

ACTIVE USE CASES

- Quantitative Single Neuron Modeling Competition
 - reduced, tractable neuron models
 - <http://github.com/incf/QSNMC>
 - tested using data collected by original authors

ACTIVE USE CASES

- Quantitative Single Neuron Modeling Competition
 - reduced, tractable neuron models
 - <http://github.com/incf/QSNMC>
 - tested using data collected by original authors
- Open Source Brain
 - biophysically detailed neuron/microcircuit models
 - <http://www.opensourcebrain.org>
 - tested using data from NeuroElectro

ACTIVE USE CASES

- Quantitative Single Neuron Modeling Competition
 - reduced, tractable neuron models
 - <http://github.com/incf/QSNMC>
 - tested using data collected by original authors
- Open Source Brain
 - biophysically detailed neuron/microcircuit models
 - <http://www.opensourcebrain.org>
 - tested using data from NeuroElectro
- OpenWorm
 - simulation of an entire organism
 - <http://www.openworm.org>
 - tested using data from neuron to behavior

Experiment Data Discovery



Experiment Data Format



Experiment Data Analysis



Data Summary



Test Instantiation



Test Score Visualization



Test Adjudication

Model Discovery



Model Description



Model Capabilities



Simulation Description



Model Interpreter

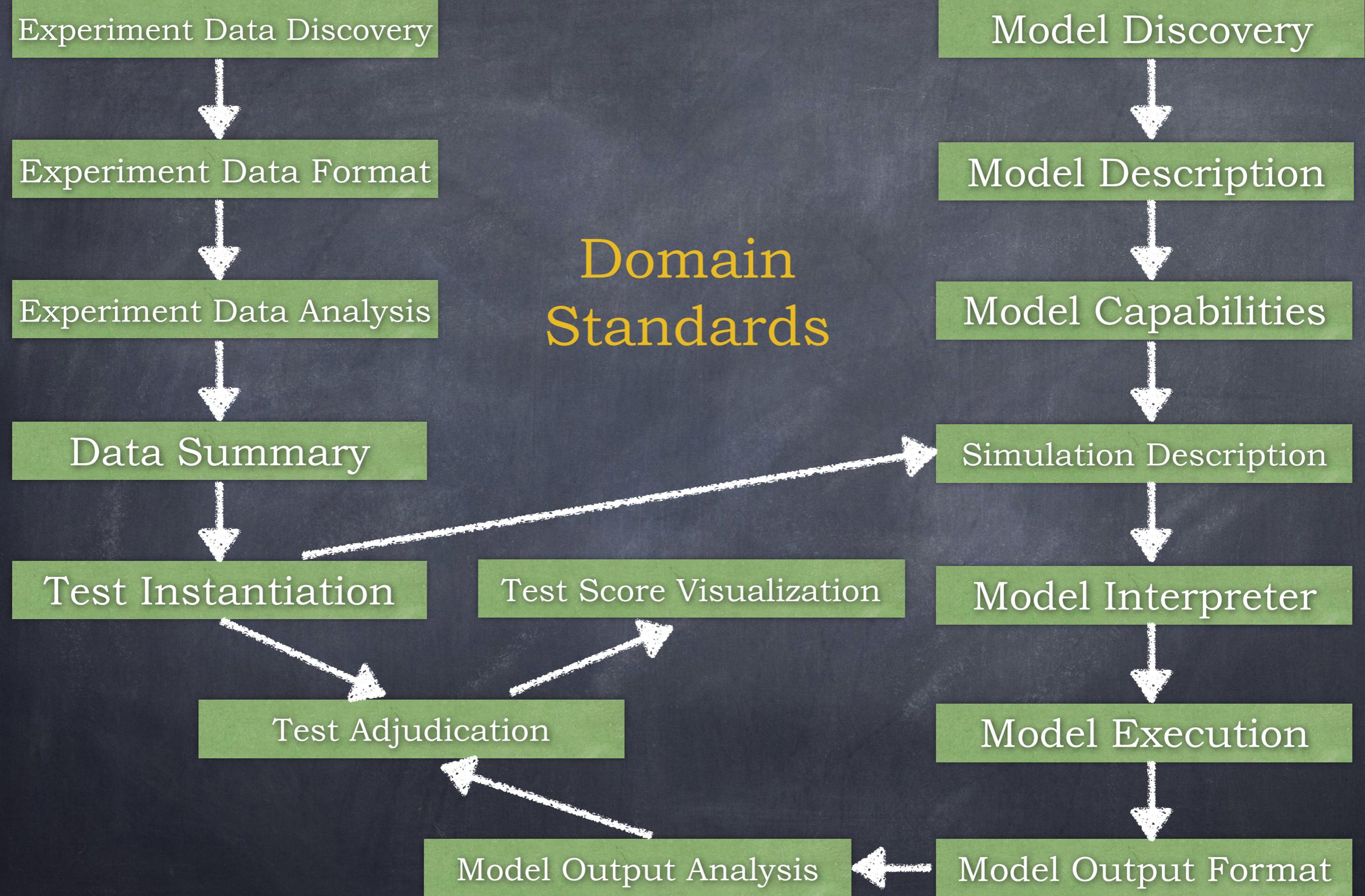


Model Execution



Model Output Format

Model Output Analysis



Experiment Data Discovery

CRCNS.org

Experiment Data Format

NIX

Experiment Data Analysis

NEO/NeuroTools

Data Summary

NeuroElectro

Test Instantiation

NeuronUnit

Test Adjudication

NumPy/SciPy

Model Output Analysis

NEO/NeuroTools

Domain Standards

Model Discovery

Open Source Brain

Model Description

NeuroML

Model Capabilities

Comp. Neuroscience Ontology

Simulation Description

SED-ML

Model Interpreter

NeuroConstruct

Model Execution

NEURON

Model Output Format

NSDF

Experiment Data Discovery

CRCNS.org

Experiment Data Format

NIX

Experiment Data Analysis

NEO/NeuroTools

Data Summary

NeuroElectro

Test Instantiation

NeuronUnit

Test Adjudication

NumPy/SciPy

Model Output Analysis

NEO/NeuroTools

Linking Domain Standards

Model Discovery

Open Source Brain

Model Description

NeuroML

Model Capabilities

Comp. Neuroscience Ontology

Simulation Description

SED-ML

Model Interpreter

NeuroConstruct

Model Execution

NEURON

Model Output Format

NSDF

NEURONUNIT

NEURONUNIT

- *SciUnit* is practical with *domain-specific* libraries for test construction and model introspection / execution that utilize domain-specific *standards*.

NEURONUNIT

- *SciUnit* is practical with *domain-specific* libraries for test construction and model introspection / execution that utilize domain-specific *standards*.
- *NeuronUnit* is a such a library for single neuron electrophysiology.

NEURONUNIT

- *SciUnit* is practical with *domain-specific* libraries for test construction and model introspection / execution that utilize domain-specific *standards*.
- *NeuronUnit* is a such a library for single neuron electrophysiology.
 - <http://github.com/scidash/neuronunit>

BENEFITS (MODELER'S PERSPECTIVE)

BENEFITS (MODELER'S PERSPECTIVE)

- Know what other models can and can't do, and what a new model (if needed) should explain in order to be better.

BENEFITS (MODELER'S PERSPECTIVE)

- Know what other models can and can't do, and what a new model (if needed) should explain in order to be better.
- The ability to continuously test your model against the data it is supposed to explain/predict.

BENEFITS (MODELER'S PERSPECTIVE)

- Know what other models can and can't do, and what a new model (if needed) should explain in order to be better.
- The ability to continuously test your model against the data it is supposed to explain/predict.
 - Accelerates model development (towards some goal of realism).

BENEFITS (MODELER'S PERSPECTIVE)

- Know what other models can and can't do, and what a new model (if needed) should explain in order to be better.
- The ability to continuously test your model against the data it is supposed to explain/predict.
 - Accelerates model development (towards some goal of realism).
 - Gives you bragging rights in the arena of model competition.

BENEFITS (MODELER'S PERSPECTIVE)

- Know what other models can and can't do, and what a new model (if needed) should explain in order to be better.
- The ability to continuously test your model against the data it is supposed to explain/predict.
 - Accelerates model development (towards some goal of realism).
 - Gives you bragging rights in the arena of model competition.
 - Address reviewers (me) who demand that your model pass SciUnit tests.

BENEFITS (MODELER'S PERSPECTIVE)

- Know what other models can and can't do, and what a new model (if needed) should explain in order to be better.
- The ability to continuously test your model against the data it is supposed to explain/predict.
 - Accelerates model development (towards some goal of realism).
 - Gives you bragging rights in the arena of model competition.
 - Address reviewers (me) who demand that your model pass SciUnit tests.
 - Post-publication review of your model, *even as new data come to light.*

BENEFITS (MODELER'S PERSPECTIVE)

- Know what other models can and can't do, and what a new model (if needed) should explain in order to be better.
- The ability to continuously test your model against the data it is supposed to explain/predict.
 - Accelerates model development (towards some goal of realism).
 - Gives you bragging rights in the arena of model competition.
 - Address reviewers (me) who demand that your model pass SciUnit tests.
 - Post-publication review of your model, *even as new data come to light.*
 - Look your child in the face when they ask if you, an alleged scientist, used the scientific method in the development of your model.

BENEFITS

(EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

BENEFITS

(EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

- The ability to put your observations in context.

BENEFITS

(EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

- The ability to put your observations in context.
 - Is there a model that explains my data?

BENEFITS

(EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

- The ability to put your observations in context.
 - Is there a model that explains my data?
 - Which model best explains my data?

BENEFITS

(EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

- The ability to put your observations in context.
 - Is there a model that explains my data?
 - Which model best explains my data?
 - What other data does it explain?

BENEFITS

(EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

- The ability to put your observations in context.
 - Is there a model that explains my data?
 - Which model best explains my data?
 - What other data does it explain?
- Pre-experiment, grant stage discovery of hypothesis implications.

BENEFITS

(EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

- The ability to put your observations in context.
 - Is there a model that explains my data?
 - Which model best explains my data?
 - What other data does it explain?
- Pre-experiment, grant stage discovery of hypothesis implications.
 - If I do experiment E and get result Y, it will support model A.

BENEFITS

(EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

- The ability to put your observations in context.
 - Is there a model that explains my data?
 - Which model best explains my data?
 - What other data does it explain?
- Pre-experiment, grant stage discovery of hypothesis implications.
 - If I do experiment E and get result Y, it will support model A.
 - If I do experiment E and get result Z, it will support model B.

BENEFITS

(EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

- The ability to put your observations in context.
 - Is there a model that explains my data?
 - Which model best explains my data?
 - What other data does it explain?
- Pre-experiment, grant stage discovery of hypothesis implications.
 - If I do experiment E and get result Y, it will support model A.
 - If I do experiment E and get result Z, it will support model B.
- Increased community awareness of the data you collected.

BENEFITS

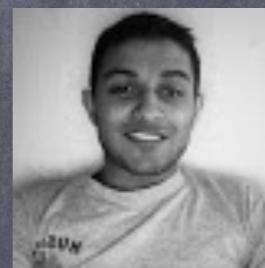
(EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

- The ability to put your observations in context.
 - Is there a model that explains my data?
 - Which model best explains my data?
 - What other data does it explain?
- Pre-experiment, grant stage discovery of hypothesis implications.
 - If I do experiment E and get result Y, it will support model A.
 - If I do experiment E and get result Z, it will support model B.
- Increased community awareness of the data you collected.
 - It could become the gold standard by which models are judged!

ACKNOWLEDGEMENTS

- **Cyrus Omar**

Carnegie Mellon Univ., USA



- **Shreejoy Tripathy**

Univ. of British Columbia, Canada



- **Sharon Crook**

Arizona State University, USA



- **Padraig Gleeson**

University College, London, UK



Appendix

SCIUNIT: CAPABILITIES

SCIUNIT: CAPABILITIES

- A *capability* is a guarantee that the model implements certain methods.

SCIUNIT: CAPABILITIES

- A *capability* is a guarantee that the model implements certain methods.

SCIUNIT: CAPABILITIES

- A *capability* is a guarantee that the model implements certain methods.
- Each *model* declares its capabilities via inheritance, and satisfies them via implementation.

SCIUNIT: CAPABILITIES

- A ***capability*** is a guarantee that the model implements certain methods.
- Each ***model*** declares its capabilities via inheritance, and satisfies them via implementation.
- Each ***test*** lists its *required capabilities*.

SCIUNIT: CAPABILITIES

SCIUNIT: CAPABILITIES

```
class ProducesSpikes(Capability):
    """
    Indicates that the model produces spikes.
    No duration is required for these spikes.
    """

    def get_spikes(self):
        """
        Gets computed spike times from the model.

        Arguments: None.
        Returns: a NeuroTools SpikeTrain object.
        """

        raise NotImplementedError()
```

SCIUNIT: CAPABILITIES

```
class ProducesSpikes(Capability):
    """
    Indicates that the model produces spikes.
    No duration is required for these spikes.
    """

    def get_spikes(self):
        """
        Gets computed spike times from the model.

        Arguments: None.
        Returns: a NeuroTools SpikeTrain object.
        """

        raise NotImplementedError()
```

- Every capability inherits from `sciunit.Capability`

SCIUNIT: CAPABILITIES

```
class ProducesSpikes(Capability):
    """
    Indicates that the model produces spikes.
    No duration is required for these spikes.
    """

    def get_spikes(self):
        """
        Gets computed spike times from the model.

        Arguments: None.
        Returns: a NeuroTools SpikeTrain object.
        """

        raise NotImplementedError()
```

- Every capability inherits from `sciunit.Capability`
- Every capability is designed by a *community* of modelers and testers.

SCIUNIT: CAPABILITIES

```
class ProducesSpikes(Capability):
    """
    Indicates that the model produces spikes.
    No duration is required for these spikes.
    """

    def get_spikes(self):
        """
        Gets computed spike times from the model.

        Arguments: None.
        Returns: a NeuroTools SpikeTrain object.
        """

        raise NotImplementedError()
```

- Every capability inherits from `sciunit.Capability`
- Every capability is designed by a *community* of modelers and testers.
- Capabilities are *implemented* by those who understand the model being tested.

SCIUNIT: TESTS

SCIUNIT: TESTS

```
class SpikeWidthTest(sciunit.Test):
    ...
    def validate_observation(self, observation):
        try:
            assert type(observation['mean']) is float
            assert type(observation['std']) is float
        except Exception,e:
            raise ObservationError("Observation must be of the form \
                {'mean':float,'std':float}")
    def generate_prediction(self, model):
        """Implementation of sciunit.Test.generate_prediction."""
        # Method implementation guaranteed by ProducesSpikes capability.
        widths = model.get_spike_widths()
        # Put prediction in a form that compute_score() can use.
        prediction = {'mean':np.mean(widths),
                      'std':np.std(widths)}
        return prediction
    def compute_score(self, observation, prediction):
        """Implementation of sciunit.Test.score_prediction."""
        score = sciunit.utils.analysis.zscore(observation,prediction)
        return sciunit.scores.ZScore(score)
```

SCIUNIT: TESTS

```
class SpikeWidthTest(sciunit.Test):  
    ...  
  
    def validate_observation(self, observation):  
        try:  
            assert type(observation['mean']) is float  
            assert type(observation['std']) is float  
        except Exception,e:  
            raise ObservationError("Observation must be of the form \\\n                {'mean':float,'std':float}")  
  
    def generate_prediction(self, model):  
        """Implementation of sciunit.Test.generate_prediction."""  
        # Method implementation guaranteed by ProducesSpikes capability.  
        widths = model.get_spike_widths()  
        # Put prediction in a form that compute_score() can use.  
        prediction = {'mean':np.mean(widths),  
                      'std':np.std(widths)}  
        return prediction  
  
    def compute_score(self, observation, prediction):  
        """Implementation of sciunit.Test.score_prediction."""  
        score = sciunit.utils.analysis.zscore(observation,prediction)  
        return sciunit.scores.ZScore(score)
```

- Check the observation.

SCIUNIT: TESTS

```
class SpikeWidthTest(sciunit.Test):  
    ...  
  
    def validate_observation(self, observation):  
        try:  
            assert type(observation['mean']) is float  
            assert type(observation['std']) is float  
        except Exception,e:  
            raise ObservationError("Observation must be of the form \\\n                {'mean':float,'std':float}")  
  
    def generate_prediction(self, model):  
        """Implementation of sciunit.Test.generate_prediction."""  
        # Method implementation guaranteed by ProducesSpikes capability.  
        widths = model.get_spike_widths()  
        # Put prediction in a form that compute_score() can use.  
        prediction = {'mean':np.mean(widths),  
                      'std':np.std(widths)}  
        return prediction  
  
    def compute_score(self, observation, prediction):  
        """Implementation of sciunit.Test.score_prediction."""  
        score = sciunit.utils.analysis.zscore(observation,prediction)  
        return sciunit.scores.ZScore(score)
```

- Check the observation.

SCIUNIT: TESTS

```
class SpikeWidthTest(sciunit.Test):  
    ...  
  
    def validate_observation(self, observation):  
        try:  
            assert type(observation['mean']) is float  
            assert type(observation['std']) is float  
        except Exception,e:  
            raise ObservationError("Observation must be of the form \\\n                {'mean':float,'std':float}")  
  
    def generate_prediction(self, model):  
        """Implementation of sciunit.Test.generate_prediction."""  
        # Method implementation guaranteed by ProducesSpikes capability.  
        widths = model.get_spike_widths()  
        # Put prediction in a form that compute_score() can use.  
        prediction = {'mean':np.mean(widths),  
                      'std':np.std(widths)}  
        return prediction  
  
    def compute_score(self, observation, prediction):  
        """Implementation of sciunit.Test.score_prediction."""  
        score = sciunit.utils.analysis.zscore(observation,prediction)  
        return sciunit.scores.ZScore(score)
```

- Check the observation.

SCIUNIT: TESTS

```
class SpikeWidthTest(sciunit.Test):  
    ...  
  
    def validate_observation(self, observation):  
        try:  
            assert type(observation['mean']) is float  
            assert type(observation['std']) is float  
        except Exception,e:  
            raise ObservationError("Observation must be of the form \\\n                {'mean':float,'std':float}")  
  
    def generate_prediction(self, model):  
        """Implementation of sciunit.Test.generate_prediction."""  
        # Method implementation guaranteed by ProducesSpikes capability.  
        widths = model.get_spike_widths()  
        # Put prediction in a form that compute_score() can use.  
        prediction = {'mean':np.mean(widths),  
                      'std':np.std(widths)}  
        return prediction  
  
    def compute_score(self, observation, prediction):  
        """Implementation of sciunit.Test.score_prediction."""  
        score = sciunit.utils.analysis.zscore(observation,prediction)  
        return sciunit.scores.ZScore(score)
```

- Check the observation.

SCIUNIT: TESTS

```
class SpikeWidthTest(sciunit.Test):  
    ...  
  
    def validate_observation(self, observation):  
        try:  
            assert type(observation['mean']) is float  
            assert type(observation['std']) is float  
        except Exception,e:  
            raise ObservationError("Observation must be of the form \\\n            {'mean':float,'std':float}")  
  
    def generate_prediction(self, model):  
        """Implementation of sciunit.Test.generate_prediction.""""  
        # Method implementation guaranteed by ProducesSpikes capability.  
        widths = model.get_spike_widths()  
        # Put prediction in a form that compute_score() can use.  
        prediction = {'mean':np.mean(widths),  
                      'std':np.std(widths)}  
        return prediction  
  
    def compute_score(self, observation, prediction):  
        """Implementation of sciunit.Test.score_prediction.""""  
        score = sciunit.utils.analysis.zscore(observation,prediction)  
        return sciunit.scores.ZScore(score)
```

- Check the observation.

SCIUNIT: TESTS

```
class SpikeWidthTest(sciunit.Test):  
    ...  
  
    def validate_observation(self, observation):  
        try:  
            assert type(observation['mean']) is float  
            assert type(observation['std']) is float  
        except Exception,e:  
            raise ObservationError("Observation must be of the form \\\n                {'mean':float,'std':float}")  
  
    def generate_prediction(self, model):  
        """Implementation of sciunit.Test.generate_prediction."""  
        # Method implementation guaranteed by ProducesSpikes capability.  
        widths = model.get_spike_widths()  
        # Put prediction in a form that compute_score() can use.  
        prediction = {'mean':np.mean(widths),  
                      'std':np.std(widths)}  
        return prediction  
  
    def compute_score(self, observation, prediction):  
        """Implementation of sciunit.Test.score_prediction."""  
        score = sciunit.utils.analysis.zscore(observation,prediction)  
        return sciunit.scores.ZScore(score)
```

- Check the observation.
- Coax a prediction from the model.

SCIUNIT: TESTS

```
class SpikeWidthTest(sciunit.Test):  
    ...  
  
    def validate_observation(self, observation):  
        try:  
            assert type(observation['mean']) is float  
            assert type(observation['std']) is float  
        except Exception,e:  
            raise ObservationError("Observation must be of the form \\\n                {'mean':float,'std':float}")  
  
    def generate_prediction(self, model):  
        """Implementation of sciunit.Test.generate_prediction."""  
        # Method implementation guaranteed by ProducesSpikes capability.  
        widths = model.get_spike_widths()  
        # Put prediction in a form that compute_score() can use.  
        prediction = {'mean':np.mean(widths),  
                      'std':np.std(widths)}  
        return prediction  
  
    def compute_score(self, observation, prediction):  
        """Implementation of sciunit.Test.score_prediction."""  
        score = sciunit.utils.analysis.zscore(observation,prediction)  
        return sciunit.scores.ZScore(score)
```

- Check the observation.
- Coax a prediction from the model.

SCIUNIT: TESTS

```
class SpikeWidthTest(sciunit.Test):  
    ...  
  
    def validate_observation(self, observation):  
        try:  
            assert type(observation['mean']) is float  
            assert type(observation['std']) is float  
        except Exception,e:  
            raise ObservationError("Observation must be of the form \\\n                {'mean':float,'std':float}")  
  
    def generate_prediction(self, model):  
        """Implementation of sciunit.Test.generate_prediction."""  
        # Method implementation guaranteed by ProducesSpikes capability.  
        widths = model.get_spike_widths()  
        # Put prediction in a form that compute_score() can use.  
        prediction = {'mean':np.mean(widths),  
                      'std':np.std(widths)}  
        return prediction  
  
    def compute_score(self, observation, prediction):  
        """Implementation of sciunit.Test.score_prediction."""  
        score = sciunit.utils.analysis.zscore(observation,prediction)  
        return sciunit.scores.ZScore(score)
```

- Check the observation.
- Coax a prediction from the model.

SCIUNIT: TESTS

```
class SpikeWidthTest(sciunit.Test):  
    ...  
  
    def validate_observation(self, observation):  
        try:  
            assert type(observation['mean']) is float  
            assert type(observation['std']) is float  
        except Exception,e:  
            raise ObservationError("Observation must be of the form \\\n                {'mean':float,'std':float}")  
  
    def generate_prediction(self, model):  
        """Implementation of sciunit.Test.generate_prediction."""  
        # Method implementation guaranteed by ProducesSpikes capability.  
        widths = model.get_spike_widths()  
        # Put prediction in a form that compute_score() can use.  
        prediction = {'mean':np.mean(widths),  
                      'std':np.std(widths)}  
        return prediction  
  
    def compute_score(self, observation, prediction):  
        """Implementation of sciunit.Test.score_prediction."""  
        score = sciunit.utils.analysis.zscore(observation,prediction)  
        return sciunit.scores.ZScore(score)
```

- Check the observation.
- Coax a prediction from the model.

SCIUNIT: TESTS

```
class SpikeWidthTest(sciunit.Test):  
    ...  
  
    def validate_observation(self, observation):  
        try:  
            assert type(observation['mean']) is float  
            assert type(observation['std']) is float  
        except Exception,e:  
            raise ObservationError("Observation must be of the form \\\n                {'mean':float,'std':float}")  
  
    def generate_prediction(self, model):  
        """Implementation of sciunit.Test.generate_prediction."""  
        # Method implementation guaranteed by ProducesSpikes capability.  
        widths = model.get_spike_widths()  
        # Put prediction in a form that compute_score() can use.  
        prediction = {'mean':np.mean(widths),  
                      'std':np.std(widths)}  
        return prediction  
  
    def compute_score(self, observation, prediction):  
        """Implementation of sciunit.Test.score_prediction."""  
        score = sciunit.utils.analysis.zscore(observation,prediction)  
        return sciunit.scores.ZScore(score)
```

- Check the observation.
- Coax a prediction from the model.
- Produce an indicator of model/data agreement.

SCIUNIT: TESTS

```
class SpikeWidthTest(sciunit.Test):  
    ...  
  
    def validate_observation(self, observation):  
        try:  
            assert type(observation['mean']) is float  
            assert type(observation['std']) is float  
        except Exception,e:  
            raise ObservationError("Observation must be of the form \\\n                {'mean':float,'std':float}")  
  
    def generate_prediction(self, model):  
        """Implementation of sciunit.Test.generate_prediction."""  
        # Method implementation guaranteed by ProducesSpikes capability.  
        widths = model.get_spike_widths()  
        # Put prediction in a form that compute_score() can use.  
        prediction = {'mean':np.mean(widths),  
                      'std':np.std(widths)}  
        return prediction  
  
    def compute_score(self, observation, prediction):  
        """Implementation of sciunit.Test.score_prediction."""  
        score = sciunit.utils.analysis.zscore(observation,prediction)  
        return sciunit.scores.ZScore(score)
```

- The *sciunit.Test* base class has a *judge* method which takes a model, checks capabilities, and invokes all of the above to return a score.

- Check the observation.
- Coax a prediction from the model.
- Produce an indicator of model/data agreement.

SCIUNIT: MODELS

SCIUNIT: MODELS

```
class LIFModel(sciunit.Model,ProducesMembranePotential,ProducesSpikes):  
    """A leaky integrate and fire model."""  
  
    def get_spikes(self):  
        # Get membrane potential.  
        membrane_potential = self.get_membrane_potential()  
        # Extract spike times.  
        spike_times = membrane_potential.threshold(0)  
        return spike_times  
  
    def get_membrane_potential(self):  
        # Create membrane potential object from model attribute.  
        membrane_potential = AnalogSignal(self.v_m, self.dt)  
        return membrane_potential  
  
    def simulate(self,...):  
        # Sets v_m, etc.  
        ...
```

SCIUNIT: MODELS

```
class LIFModel(sciunit.Model,ProducesMembranePotential,ProducesSpikes):
    """A leaky integrate and fire model."""

    def get_spikes(self):
        # Get membrane potential.
        membrane_potential = self.get_membrane_potential()
        # Extract spike times.
        spike_times = membrane_potential.threshold(0)
        return spike_times

    def get_membrane_potential(self):
        # Create membrane potential object from model attribute.
        membrane_potential = AnalogSignal(self.v_m, self.dt)
        return membrane_potential

    def simulate(self, ...):
        # Sets v_m, etc.
        ...
```

- *SciUnit* models declare their capabilities via *inheritance*.

SCIUNIT: MODELS

```
class LIFModel(sciunit.Model,ProducesMembranePotential,ProducesSpikes):
    """A leaky integrate and fire model."""

    def get_spikes(self):
        # Get membrane potential.
        membrane_potential = self.get_membrane_potential()
        # Extract spike times.
        spike_times = membrane_potential.threshold(0)
        return spike_times

    def get_membrane_potential(self):
        # Create membrane potential object from model attribute.
        membrane_potential = AnalogSignal(self.v_m, self.dt)
        return membrane_potential

    def simulate(self,...):
        # Sets v_m, etc.
        ...
```

- *SciUnit* models declare their capabilities via *inheritance*.
- *SciUnit* models *implement* capabilities to match the design of the underlying real model.

SCIUNIT: CAPABILITIES

SCIUNIT: CAPABILITIES

```
class ProducesActionPotentials(ProducesSpikes):  
    """Indicates the model produces action potential waveforms.  
    Waveforms must have a temporal extent.  
    """  
  
    def get_action_potentials(self):  
        """Gets action potential waveform chunks from the model.  
  
        Returns  
        -----  
        NeuroTools.signals.AnalogSignalList  
            A list of spike waveforms  
        """  
  
        raise NotImplementedError()
```

SCIUNIT: CAPABILITIES

```
class ProducesActionPotentials(ProducesSpikes):  
    """Indicates the model produces action potential waveforms.  
    Waveforms must have a temporal extent.  
    """  
  
    def get_action_potentials(self):  
        """Gets action potential waveform chunks from the model.  
  
        Returns  
        -----  
        NeuroTools.signals.AnalogSignalList  
            A list of spike waveforms  
        """  
  
        raise NotImplementedError()
```

- Capabilities can be built up from other capabilities via inheritance.

SCIUNIT: CAPABILITIES

```
class ProducesActionPotentials(ProducesSpikes):  
    """Indicates the model produces action potential waveforms.  
    Waveforms must have a temporal extent.  
    """  
  
    def get_action_potentials(self):  
        """Gets action potential waveform chunks from the model.  
  
        Returns  
        -----  
        NeuroTools.signals.AnalogSignalList  
            A list of spike waveforms  
        """  
  
        raise NotImplementedError()
```

- Capabilities can be built up from other capabilities via inheritance.

SCIUNIT: CAPABILITIES

```
class ProducesActionPotentials(ProducesSpikes):  
    """Indicates the model produces action potential waveforms.  
    Waveforms must have a temporal extent.  
    """  
  
    def get_action_potentials(self):  
        """Gets action potential waveform chunks from the model.  
  
        Returns  
        -----  
        NeuroTools.signals.AnalogSignalList  
            A list of spike waveforms  
        """  
  
        raise NotImplementedError()
```

- Capabilities can be built up from other capabilities via inheritance.
- Previously unconsidered capabilities can be added easily.

SCIUNIT: TESTS

SCIUNIT: TESTS

- A ***test*** class is a subclass of *sciunit.Test*

SCIUNIT: TESTS

- A ***test*** class is a subclass of *sciunit.Test*
- Interacts with models *only through* capabilities, in order to extract a *prediction*.

SCIUNIT: TESTS

- A ***test*** class is a subclass of *sciunit.Test*
- Interacts with models *only through* capabilities, in order to extract a *prediction*.
- Compares that *prediction* with *observed data*.

SCIUNIT: TESTS

- A ***test*** class is a subclass of *sciunit.Test*
- Interacts with models *only through* capabilities, in order to extract a *prediction*.
- Compares that *prediction* with *observed data*.
- Returns a score indicating agreement.

SCIUNIT: TESTS

- A **test** class is a subclass of `sciunit.Test`
 - Interacts with models *only through* capabilities, in order to extract a *prediction*.
 - Compares that *prediction* with *observed data*.
 - Returns a score indicating agreement.
- A specific test is an instance of that class, parameterized by the *observed data*.

SCIUNIT: TESTS

- A **test** class is a subclass of `sciunit.Test`
 - Interacts with models *only through* capabilities, in order to extract a *prediction*.
 - Compares that *prediction* with *observed data*.
 - Returns a score indicating agreement.
- A specific test is an instance of that class, parameterized by the *observed data*.
- Tests are inherently **subjective**.

NEURONUNIT: MODELS

NEURONUNIT: MODELS

- What kinds of models might people want to test?

NEURONUNIT: MODELS

- What kinds of models might people want to test?
- How can we maximally reuse *SciUnit* model classes to apply to the largest number of actual models?

NEURONUNIT: MODELS

- What kinds of models might people want to test?
- How can we maximally reuse *SciUnit* model classes to apply to the largest number of actual models?
- How can we support such a large number of model languages and simulation environments?

NEURONUNIT: MODELS

- What kinds of models might people want to test?
- How can we maximally reuse *SciUnit* model classes to apply to the largest number of actual models?
- How can we support such a large number of model languages and simulation environments?

NEURONUNIT: MODELS

- What kinds of models might people want to test?
- How can we maximally reuse *SciUnit* model classes to apply to the largest number of actual models?
- How can we support such a large number of model languages and simulation environments?
- **Standards:**

NEURONUNIT: MODELS

- What kinds of models might people want to test?
- How can we maximally reuse *SciUnit* model classes to apply to the largest number of actual models?
- How can we support such a large number of model languages and simulation environments?
- **Standards:**
 - Model description: *NeuroML* (<http://neuroml.org>)

NEURONUNIT: MODELS

- What kinds of models might people want to test?
- How can we maximally reuse *SciUnit* model classes to apply to the largest number of actual models?
- How can we support such a large number of model languages and simulation environments?
- **Standards:**
 - Model description: *NeuroML* (<http://neuroml.org>)
 - Simulation description: *SED-ML* (<http://sed-ml.org>)

NEURONUNIT: MODELS

- What kinds of models might people want to test?
- How can we maximally reuse *SciUnit* model classes to apply to the largest number of actual models?
- How can we support such a large number of model languages and simulation environments?
- **Standards:**
 - Model description: *NeuroML* (<http://neuroml.org>)
 - Simulation description: *SED-ML* (<http://sed-ml.org>)
 - Simulation execution: ... whatever reads NeuroML (neuroConstruct, NEURON, etc.)

NEURONUNIT: MODELS

NEURONUNIT: MODELS

- An example model from Open Source Brain ([http://
www.opensourcebrain.org](http://www.opensourcebrain.org))

NEURONUNIT: MODELS

- An example model from Open Source Brain (<http://www.opensourcebrain.org>)
 - Model description given in *NeuroML*.

NEURONUNIT: MODELS

- An example model from Open Source Brain (<http://www.opensourcebrain.org>)
 - Model description given in *NeuroML*.
 - Simulation description given in *neuroConstruct*.

NEURONUNIT: MODELS

- An example model from Open Source Brain (<http://www.opensourcebrain.org>)
 - Model description given in *NeuroML*.
 - Simulation description given in *neuroConstruct*.
 - Recapitulates (*De Schutter and Bower, 1994*).

NEURONUNIT: MODELS

- An example model from Open Source Brain (<http://www.opensourcebrain.org>)
 - Model description given in *NeuroML*.
 - Simulation description given in *neuroConstruct*.
 - Recapitulates (*De Schutter and Bower, 1994*).

```
from neuronunit.models import OSBModel

# Purkinje Cell model from OSB at cerebellum/cerebellar_purkinje_cell/PurkinjeCell"""
PurkinjeCell = OSBModel.make_model("cerebellum",
                                    "cerebellar_purkinje_cell",
                                    "PurkinjeCell")

# Initialize (parameterize) the model with some initialization parameters.
model = PurkinjeCell(population_name="OrigChansCellGroup_0")
```

NEURONUNIT: MODELS

- An example model from Open Source Brain (<http://www.opensourcebrain.org>)
 - Model description given in *NeuroML*.
 - Simulation description given in *neuroConstruct*.
 - Recapitulates (*De Schutter and Bower, 1994*).

```
from neuronunit.models import OSBModel

# Purkinje Cell model from OSB at cerebellum/cerebellar_purkinje_cell/PurkinjeCell"""
PurkinjeCell = OSBModel.make_model("cerebellum",
                                    "cerebellar_purkinje_cell",
                                    "PurkinjeCell")

# Initialize (parameterize) the model with some initialization parameters.
model = PurkinjeCell(population_name="OrigChansCellGroup_0")
```

... followed by many interactions with this model via its capabilities, in the course of test execution.

SCIUNIT: TEST SUITES

SCIUNIT: TEST SUITES

- *Test suites* are collections of tests organized around a theme.

SCIUNIT: TEST SUITES

- *Test suites* are collections of tests organized around a theme.
 - Each test examines one aspect of the model's validity.

SCIUNIT: TEST SUITES

- *Test suites* are collections of tests organized around a theme.
 - Each test examines one aspect of the model's validity.
 - e.g. each test could examine the response of a neuron model to one amplitude or waveform of injected current.

SCIUNIT: TEST SUITES

- *Test suites* are collections of tests organized around a theme.
 - Each test examines one aspect of the model's validity.
 - e.g. each test could examine the response of a neuron model to one amplitude or waveform of injected current.
 - i.e. each test could correspond to the same test class, but instantiated with different parameter sets.

SCIUNIT: TEST SUITES

- *Test suites* are collections of tests organized around a theme.
 - Each test examines one aspect of the model's validity.
 - e.g. each test could examine the response of a neuron model to one amplitude or waveform of injected current.
 - i.e. each test could correspond to the same test class, but instantiated with different parameter sets.

```
class InjectedCurrentSpikeWidthTest(SpikeWidthTest):  
    """Tests the full widths of spikes at their half-maximum under current injection."""  
  
    def __init__(self,  
                 observation={'mean':None,'std':None},  
                 name="Action potential width under current injection",  
                 params={'injected_current':{'ampl':0.0}}):  
        """Takes a steady-state current to be injected into a cell."""  
  
    ...
```

SCIUNIT: TEST SUITES

- *Test suites* are collections of tests organized around a theme.
 - Each test examines one aspect of the model's validity.
 - e.g. each test could examine the response of a neuron model to one amplitude or waveform of injected current.
 - i.e. each test could correspond to the same test class, but instantiated with different parameter sets.

```
class InjectedCurrentSpikeWidthTest(SpikeWidthTest):  
    """Tests the full widths of spikes at their half-maximum under current injection."""  
  
    def __init__(self,  
                 observation={'mean':None,'std':None},  
                 name="Action potential width under current injection",  
                 params={'injected_current':{'ampl':0.0}}):  
        """Takes a steady-state current to be injected into a cell."""  
  
    ...
```

```
def injection_params(amplitude):  
    return {'injected_current':{'ampl':amplitude}}:  
  
width_test_1 = InjectedCurrentSpikeWidthTest(observation, injection_params(25.0))  
width_test_2 = InjectedCurrentSpikeWidthTest(observation, injection_params(50.0))  
width_test_3 = InjectedCurrentSpikeWidthTest(observation, injection_params(75.0))  
  
width_suite = sciunit.TestSuite([width_test_1,width_test_2,width_test_3])
```

NEURONUNIT

NEURONUNIT

- SciUnit *capability* classes crafted to span the things one would ask single cell neurophysiology models to do.

NEURONUNIT

- SciUnit *capability* classes crafted to span the things one would ask single cell neurophysiology models to do.
 - *Helper functions* for implementation of these capabilities to match common model/simulation platforms (*neuroConstruct*)

NEURONUNIT

- SciUnit *capability* classes crafted to span the things one would ask single cell neurophysiology models to do.
 - *Helper functions* for implementation of these capabilities to match common model/simulation platforms (*neuroConstruct*)
- SciUnit *model* classes that implement these capabilities as well as platform-specific model instantiation.

NEURONUNIT

- SciUnit *capability* classes crafted to span the things one would ask single cell neurophysiology models to do.
 - *Helper functions* for implementation of these capabilities to match common model/simulation platforms (*neuroConstruct*)
- SciUnit *model* classes that implement these capabilities as well as platform-specific model instantiation.
- SciUnit *test* classes that can be initialized with human-readable observation metadata.

NEURONUNIT

- SciUnit *capability* classes crafted to span the things one would ask single cell neurophysiology models to do.
 - *Helper functions* for implementation of these capabilities to match common model/simulation platforms (*neuroConstruct*)
- SciUnit *model* classes that implement these capabilities as well as platform-specific model instantiation.
- SciUnit *test* classes that can be initialized with human-readable observation metadata.
 - Observed data for test parameterization retrieved automatically from trusted data repositories.

NEURONUNIT: CAPABILITIES

NEURONUNIT: CAPABILITIES

- What capabilities might single neuron models have?

NEURONUNIT: CAPABILITIES

- What capabilities might single neuron models have?
- How these should be organized to reflect the functions one might call in order to interact with these models?

NEURONUNIT: CAPABILITIES

- What capabilities might single neuron models have?
- How these should be organized to reflect the functions one might call in order to interact with these models?
- Community input requested.

NEURONUNIT: CAPABILITIES

- What capabilities might single neuron models have?
- How these should be organized to reflect the functions one might call in order to interact with these models?
- Community input requested.
- Capability method arguments, return values, and provided implementations ought to reflect standards in analysis of neuron physiology data.

NEURONUNIT: CAPABILITIES

- What capabilities might single neuron models have?
- How these should be organized to reflect the functions one might call in order to interact with these models?
- Community input requested.
- Capability method arguments, return values, and provided implementations ought to reflect standards in analysis of neuron physiology data.
 - *NumPy/SciPy* (<http://www.numpy.org/>)

NEURONUNIT: CAPABILITIES

- What capabilities might single neuron models have?
- How these should be organized to reflect the functions one might call in order to interact with these models?
- Community input requested.
- Capability method arguments, return values, and provided implementations ought to reflect standards in analysis of neuron physiology data.
 - *NumPy/SciPy* (<http://www.numpy.org/>)
 - *NEO* (<http://neuralensemble.org/neo/>)

NEURONUNIT: CAPABILITIES

- What capabilities might single neuron models have?
- How these should be organized to reflect the functions one might call in order to interact with these models?
- Community input requested.
- Capability method arguments, return values, and provided implementations ought to reflect standards in analysis of neuron physiology data.
 - *NumPy/SciPy* (<http://www.numpy.org/>)
 - *NEO* (<http://neuralensemble.org/neo/>)
 - *NeuroTools* (<http://neuralensemble.org/NeuroTools/>)

NEURONUNIT: CAPABILITIES

- What capabilities might single neuron models have?
- How these should be organized to reflect the functions one might call in order to interact with these models?
- Community input requested.
- Capability method arguments, return values, and provided implementations ought to reflect standards in analysis of neuron physiology data.
 - *NumPy/SciPy* (<http://www.numpy.org/>)
 - *NEO* (<http://neuralensemble.org/neo/>)
 - *NeuroTools* (<http://neuralensemble.org/NeuroTools/>)
 - Being replaced by ElectroPhysiology Analysis Toolkit (*ElePhAnT*)

NEURONUNIT: TESTS

NEURONUNIT: TESTS

- What kinds of model predictions might a neuron modeler want to get right?

NEURONUNIT: TESTS

- What kinds of model predictions might a neuron modeler want to get right?
- Which observations can be help evaluate those predictions?

NEURONUNIT: TESTS

- What kinds of model predictions might a neuron modeler want to get right?
- Which observations can be help evaluate those predictions?
- How can we avoid hand-selecting observations?

NEURONUNIT: TESTS

- What kinds of model predictions might a neuron modeler want to get right?
- Which observations can be help evaluate those predictions?
- How can we avoid hand-selecting observations?
- SciUnit *test* classes to automate the parameterization of tests using standard repositories of curated observations.

NEURONUNIT: TESTS

- What kinds of model predictions might a neuron modeler want to get right?
- Which observations can be help evaluate those predictions?
- How can we avoid hand-selecting observations?
- SciUnit *test* classes to automate the parameterization of tests using standard repositories of curated observations.
 - Morphological: *NeuroMorpho* (<http://neuromorpho.org>)

NEURONUNIT: TESTS

- What kinds of model predictions might a neuron modeler want to get right?
- Which observations can be help evaluate those predictions?
- How can we avoid hand-selecting observations?
- SciUnit *test* classes to automate the parameterization of tests using standard repositories of curated observations.
 - Morphological: *NeuroMorpho* (<http://neuromorpho.org>)
 - Electrical (whole neuron): *NeuroElectro* (<http://neuroelectro.org>)

NEURONUNIT: TESTS

- What kinds of model predictions might a neuron modeler want to get right?
- Which observations can be help evaluate those predictions?
- How can we avoid hand-selecting observations?
- SciUnit *test* classes to automate the parameterization of tests using standard repositories of curated observations.
 - Morphological: *NeuroMorpho* (<http://neuromorpho.org>)
 - Electrical (whole neuron): *NeuroElectro* (<http://neuroelectro.org>)
 - Electrical (ion channel): *ChannelPedia* (<http://channelpedia.epfl.ch/>)

NEURONUNIT: TESTS

- What kinds of model predictions might a neuron modeler want to get right?
- Which observations can be help evaluate those predictions?
- How can we avoid hand-selecting observations?
- SciUnit *test* classes to automate the parameterization of tests using standard repositories of curated observations.
 - Morphological: *NeuroMorpho* (<http://neuromorpho.org>)
 - Electrical (whole neuron): *NeuroElectro* (<http://neuroelectro.org>)
 - Electrical (ion channel): *ChannelPedia* (<http://channelpedia.epfl.ch/>)
 - Synaptic: ?

VISUALIZING VALIDITY

JOURNAL OF GEOPHYSICAL RESEARCH, VOL. 104, NO. A10, PAGES 22,375–22,388, OCTOBER 1, 1999

A synthesis of solar cycle prediction techniques

David H. Hathaway, Robert M. Wilson, and Edwin J. Reichmann

NASA Marshall Space Flight Center, Huntsville, Alabama

VISUALIZING VALIDITY

JOURNAL OF GEOPHYSICAL RESEARCH, VOL. 104, NO. A10, PAGES 22,375–22,388, OCTOBER 1, 1999

A synthesis of solar cycle prediction techniques

David H. Hathaway, Robert M. Wilson, and Edwin J. Reichmann

NASA Marshall Space Flight Center, Huntsville, Alabama

```
TestSuite([SunspotTest(cycle_data) for cycle_data in all_cycle_data])
.judge([OhlsMethod, FeynmansMethod, ThompsonsMethod]).view()
```

Model	SunspotTest(19)	SunspotTest(20)	SunspotTest(21)	SunspotTest(22)
OhlsMethod	-55.4	19.1	21.8	4.4
FeynmansMethod	-42.8	9.6	26.9	3.6
ThompsonsonMethod	-17.8	8.7	-26.5	-13.6

VISUALIZING VALIDITY

```
TestSuite([SunspotTest(cycle_data) for cycle_data in all_cycle_data])  
.judge([OhlsMethod, FeynmansMethod, ThompsonsMethod]).view()
```

Model	SunspotTest(19)	SunspotTest(20)	SunspotTest(21)	SunspotTest(22)
OhlsMethod	-55.4	19.1	21.8	4.4
FeynmansMethod	-42.8	9.6	26.9	3.6
ThompsonsonMethod	-17.8	8.7	-26.5	-13.6

VISUALIZING VALIDITY

```
TestSuite([SunspotTest(cycle_data) for cycle_data in all_cycle_data])  
.judge([OhlsMethod, FeynmansMethod, ThompsonsMethod]).view()
```

Model	SunspotTest(19)	SunspotTest(20)	SunspotTest(21)	SunspotTest(22)
OhlsMethod	-55.4	19.1	21.8	4.4
FeynmansMethod	-42.8	9.6	26.9	3.6
ThompsonsonMethod	-17.8	8.7	-26.5	-13.6

- When someone develops a new model (or technique), we add a row to the validation table.

VISUALIZING VALIDITY

```
TestSuite([SunspotTest(cycle_data) for cycle_data in all_cycle_data])  
.judge([OhlsMethod, FeynmansMethod, ThompsonsMethod]).view()
```

Model	SunspotTest(19)	SunspotTest(20)	SunspotTest(21)	SunspotTest(22)
OhlsMethod	-55.4	19.1	21.8	4.4
FeynmansMethod	-42.8	9.6	26.9	3.6
ThompsonsonMethod	-17.8	8.7	-26.5	-13.6

- When someone develops a new model (or technique), we add a row to the validation table.
- Everyone immediately sees where it stands with respect to all previously test-encoded data!

VISUALIZING VALIDITY

```
TestSuite([SunspotTest(cycle_data) for cycle_data in all_cycle_data])  
.judge([OhlsMethod, FeynmansMethod, ThompsonsMethod]).view()
```

Model	SunspotTest(19)	SunspotTest(20)	SunspotTest(21)	SunspotTest(22)
OhlsMethod	-55.4	19.1	21.8	4.4
FeynmansMethod	-42.8	9.6	26.9	3.6
ThompsonsonMethod	-17.8	8.7	-26.5	-13.6

VISUALIZING VALIDITY

```
TestSuite([SunspotTest(cycle_data) for cycle_data in all_cycle_data])  
.judge([OhlsMethod, FeynmansMethod, ThompsonsMethod]).view()
```

Model	SunspotTest(19)	SunspotTest(20)	SunspotTest(21)	SunspotTest(22)
OhlsMethod	-55.4	19.1	21.8	4.4
FeynmansMethod	-42.8	9.6	26.9	3.6
ThompsonsonMethod	-17.8	8.7	-26.5	-13.6

- When new experimental data is produced, we add a new column.

VISUALIZING VALIDITY

```
TestSuite([SunspotTest(cycle_data) for cycle_data in all_cycle_data])  
.judge([OhlsMethod, FeynmansMethod, ThompsonsMethod]).view()
```

Model	SunspotTest(19)	SunspotTest(20)	SunspotTest(21)	SunspotTest(22)
OhlsMethod	-55.4	19.1	21.8	4.4
FeynmansMethod	-42.8	9.6	26.9	3.6
ThompsonsonMethod	-17.8	8.7	-26.5	-13.6

- When new experimental data is produced, we add a new column.
- Everyone immediately sees how valid all existing models are against it!

COMPETITION

COMPETITION

This branch is 14 commits ahead of scidash:master		
		Pull Request
2009a test now runs against hand-coded LIF model		
 rgerkin	authored 7 days ago	latest commit e73962fd32
 capabilities	2009a test now runs against hand-coded LIF model	7 days ago
 comparators	2009a test now runs against hand-coded LIF model	7 days ago
 docs	Added documentation for spike train similarity metric	23 days ago
 models	2009a test now runs against hand-coded LIF model	7 days ago
 records	Initial commit	a year ago
 suites	Year 2009 Test A now works for simple models	23 days ago
 tests	2009a test now runs against hand-coded LIF model	7 days ago
 .gitignore	Ignoring pickled files (too big for GitHub)	21 days ago
 README.md	Update README.md	4 months ago
 __init__.py	Made into module	23 days ago
 __main__.py	2009a test now runs against hand-coded LIF model	7 days ago

We provide a template for a
suite repository
forked from
github.com/scidash/scidash.

COMPETITION

This branch is 14 commits ahead of scidash:master		
		Pull Request
2009a test now runs against hand-coded LIF model		
 rgerkin	authored 7 days ago	latest commit e73962fd32
 capabilities	2009a test now runs against hand-coded LIF model	7 days ago
 comparators	2009a test now runs against hand-coded LIF model	7 days ago
 docs	Added documentation for spike train similarity metric	23 days ago
 models	2009a test now runs against hand-coded LIF model	7 days ago
 records	Initial commit	a year ago
 suites	Year 2009 Test A now works for simple models	23 days ago
 tests	2009a test now runs against hand-coded LIF model	7 days ago
 .gitignore	Ignoring pickled files (too big for GitHub)	21 days ago
 README.md	Update README.md	4 months ago
 __init__.py	Made into module	23 days ago
 __main__.py	2009a test now runs against hand-coded LIF model	7 days ago

We provide a template for a
suite repository
forked from
github.com/scidash/scidash.

COMPETITION

This branch is 14 commits ahead of scidash:master		
		Pull Request
2009a test now runs against hand-coded LIF model		
 rgerkin	authored 7 days ago	latest commit e73962fd32
 capabilities	2009a test now runs against hand-coded LIF model	7 days ago
 comparators	2009a test now runs against hand-coded LIF model	7 days ago
 docs	Added documentation for spike train similarity metric	23 days ago
 models	2009a test now runs against hand-coded LIF model	7 days ago
 records	Initial commit	a year ago
 suites	Year 2009 Test A now works for simple models	23 days ago
 tests	2009a test now runs against hand-coded LIF model	7 days ago
 .gitignore	Ignoring pickled files (too big for GitHub)	21 days ago
 README.md	Update README.md	4 months ago
 __init__.py	Made into module	23 days ago
 __main__.py	2009a test now runs against hand-coded LIF model	7 days ago

We provide a template for a
suite repository
forked from
github.com/scidash/scidash.

This enables:

COMPETITION

This branch is 14 commits ahead of scidash:master		
		Pull Request
2009a test now runs against hand-coded LIF model		
 rgerkin	authored 7 days ago	latest commit e73962fd32
 capabilities	2009a test now runs against hand-coded LIF model	7 days ago
 comparators	2009a test now runs against hand-coded LIF model	7 days ago
 docs	Added documentation for spike train similarity metric	23 days ago
 models	2009a test now runs against hand-coded LIF model	7 days ago
 records	Initial commit	a year ago
 suites	Year 2009 Test A now works for simple models	23 days ago
 tests	2009a test now runs against hand-coded LIF model	7 days ago
 .gitignore	Ignoring pickled files (too big for GitHub)	21 days ago
 README.md	Update README.md	4 months ago
 __init__.py	Made into module	23 days ago
 __main__.py	2009a test now runs against hand-coded LIF model	7 days ago

We provide a template for a
suite repository
forked from
github.com/scidash/scidash.

This enables:

- rapid implementation from a consistent skeleton

COMPETITION

This branch is 14 commits ahead of scidash:master		
		Pull Request
2009a test now runs against hand-coded LIF model		
 rgerkin	authored 7 days ago	latest commit e73962fd32
 capabilities	2009a test now runs against hand-coded LIF model	7 days ago
 comparators	2009a test now runs against hand-coded LIF model	7 days ago
 docs	Added documentation for spike train similarity metric	23 days ago
 models	2009a test now runs against hand-coded LIF model	7 days ago
 records	Initial commit	a year ago
 suites	Year 2009 Test A now works for simple models	23 days ago
 tests	2009a test now runs against hand-coded LIF model	7 days ago
 .gitignore	Ignoring pickled files (too big for GitHub)	21 days ago
 README.md	Update README.md	4 months ago
 __init__.py	Made into module	23 days ago
 __main__.py	2009a test now runs against hand-coded LIF model	7 days ago

This enables:

- rapid implementation from a consistent skeleton
- automation of test suite execution

We provide a template for a
suite repository
forked from
github.com/scidash/scidash.

COMPETITION

This branch is 14 commits ahead of scidash:master		
		Pull Request
2009a test now runs against hand-coded LIF model		
 rgerkin	authored 7 days ago	latest commit e73962fd32
 capabilities	2009a test now runs against hand-coded LIF model	7 days ago
 comparators	2009a test now runs against hand-coded LIF model	7 days ago
 docs	Added documentation for spike train similarity metric	23 days ago
 models	2009a test now runs against hand-coded LIF model	7 days ago
 records	Initial commit	a year ago
 suites	Year 2009 Test A now works for simple models	23 days ago
 tests	2009a test now runs against hand-coded LIF model	7 days ago
 .gitignore	Ignoring pickled files (too big for GitHub)	21 days ago
 README.md	Update README.md	4 months ago
 __init__.py	Made into module	23 days ago
 __main__.py	2009a test now runs against hand-coded LIF model	7 days ago

This enables:

- rapid implementation from a consistent skeleton
- automation of test suite execution
- forking for competing visions

We provide a template for a
suite repository
forked from
github.com/scidash/scidash.

WHAT YOU PUT IN (MODELER'S PERSPECTIVE)

WHAT YOU PUT IN (MODELER'S PERSPECTIVE)

- Identify the capabilities that a test requires of your model.

WHAT YOU PUT IN (MODELER'S PERSPECTIVE)

- Identify the capabilities that a test requires of your model.
 - If your model platform has a helper class (e.g. *NeuroConstructModel*) you are ready to go!

WHAT YOU PUT IN (MODELER'S PERSPECTIVE)

- Identify the capabilities that a test requires of your model.
 - If your model platform has a helper class (e.g. *NeuroConstructModel*) you are ready to go!
 - If not, implement the required capabilities (i.e. mostly just wrapping the existing functionality of your model in capability methods).

WHAT YOU PUT IN (EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

WHAT YOU PUT IN (EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

- Choose an analysis algorithm for adjudicating model/data goodness-of-fit.

WHAT YOU PUT IN (EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

- Choose an analysis algorithm for adjudicating model/data goodness-of-fit.
 - If that algorithm is provided in a *SciUnit* test library (e.g. *NeuronUnit*), import and call it.

WHAT YOU PUT IN (EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

- Choose an analysis algorithm for adjudicating model/data goodness-of-fit.
 - If that algorithm is provided in a *SciUnit* test library (e.g. *NeuronUnit*), import and call it.
 - If not, write your own.

WHAT YOU PUT IN (EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

- Choose an analysis algorithm for adjudicating model/data goodness-of-fit.
 - If that algorithm is provided in a *SciUnit* test library (e.g. *NeuronUnit*), import and call it.
 - If not, write your own.
- Identify the data that will parameterize your test.

WHAT YOU PUT IN (EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

- Choose an analysis algorithm for adjudicating model/data goodness-of-fit.
 - If that algorithm is provided in a *SciUnit* test library (e.g. *NeuronUnit*), import and call it.
 - If not, write your own.
- Identify the data that will parameterize your test.
 - If that data is stored in a repository that the test library has helper classes for, use them.

WHAT YOU PUT IN

(EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

- Choose an analysis algorithm for adjudicating model/data goodness-of-fit.
 - If that algorithm is provided in a *SciUnit* test library (e.g. *NeuronUnit*), import and call it.
 - If not, write your own.
- Identify the data that will parameterize your test.
 - If that data is stored in a repository that the test library has helper classes for, use them.
 - If not, encode your own data to pass to the test's constructor.